Microsoft Research Detours Package Overview

Detours is a library for intercepting binary functions on ARM, x86, x64, and IA64 machines. Detours is most commmonly used to intercept Win32 APIs calls within an application, such as to add debugging instrumentation. 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 placed in a *trampoline*. The address of the trampoline is placed in a *target pointer*. The detour function can either replace the target function or extend its semantics by invoking the target function as a subroutine through the target pointer to the trampoline.

Detours are inserted at execution time. The code of the target function is modified in memory, not on disk, thus enabling 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.

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 load a DLL into a new process. Once loaded into a process, the instrumentation DLL can detour any function in the process, whether in the application or the system libraries, such as the Windows APIs.

This technical overview of Detours is divided into four sections:

- Interception of Binary Functions
- Using Detours
- Payloads and DLL Import Editing
- Detouring 32-bit and 64-bit Processes

Developers new to Detours will find the it very useful to read all four technical overview sections and the Simple sample.

This documentation also contains the following information:

- Detours API reference
- Sample programs using the Detours APIs.
- Frequently Asked Questions (FAQ)

Interception of Binary Functions

The Detours library enables 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.

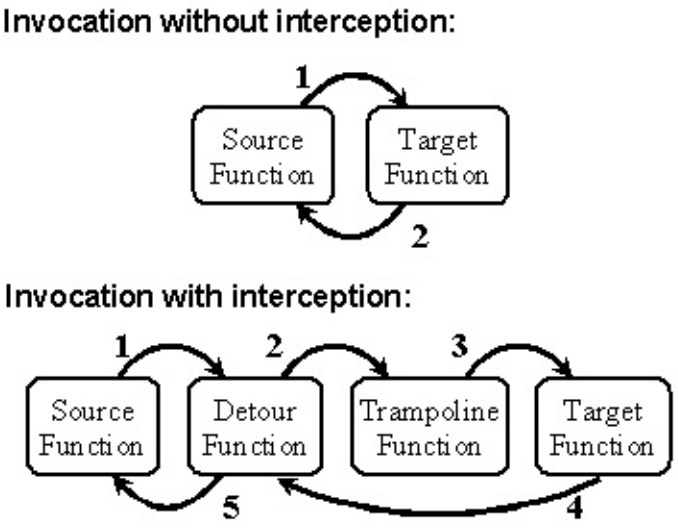

Figure 1. Control flow of invocation without Detours and with Detours.

The Detours library intercepts target functions by rewriting their in-process binary image. For each target function, Detours actually rewrites two functions,

the target function and the matching trampoline function, and one function pointer, the target pointer. The trampoline function is allocated dynamically by Detours. Prior to insertion of a detour, the trampoline contains only a single jump instruction to the target function. After insertion, the trampoline contains the initial instructions from the target function and a jump to the remainder of the target function.

The *target pointer* is initialized by the user to point to the target function. After a detour is attached to the target function, the target pointer is modified to point to the trampoline function. After the detour is detached from the target function, the target pointer is returned to point to the original target function.

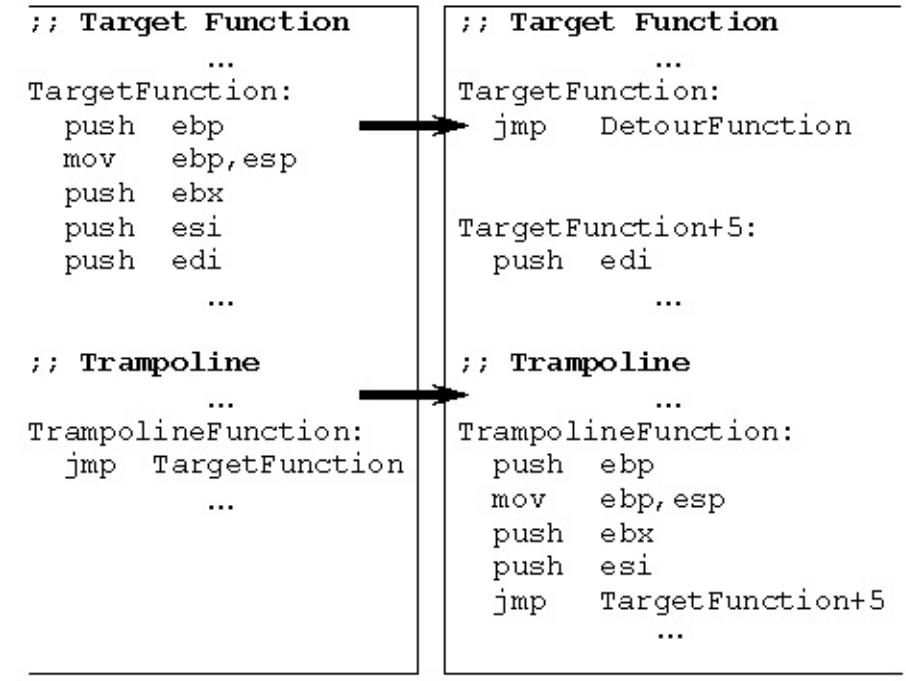

Figure 2. Trampoline and target functions, before (on the left) and after (on the right) insertion of the detour.

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 the `FlushInstructionCache` API.

Using Detours

Two things are necessary in order to detour a target function: a target pointer containing the address of the target function and a detour function. For proper interception the target function, detour function, and the target pointer must have exactly the same call signature including number of arguments and calling convention. 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

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>

static LONG dwSlept = 0;

// Target pointer for the uninstrumented Sleep API.
//
static VOID (WINAPI * TrueSleep)(DWORD dwMilliseconds) = Sleep;

// Detour function that replaces the Sleep API.
//
VOID WINAPI TimedSleep(DWORD dwMilliseconds)
{
    // Save the before and after times around calling the Sleep AP
    DWORD dwBeg = GetTickCount();
    TrueSleep(dwMilliseconds);
    DWORD dwEnd = GetTickCount();

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

// DllMain function attaches and detaches the TimedSleep detour to
// Sleep target function.  The Sleep target function is referred t
// through the TrueSleep target pointer.
//
BOOL WINAPI DllMain(HINSTANCE hinst, DWORD dwReason, LPVOID reserv
{
    if (DetourIsHelperProcess()) {
        return TRUE;
```

```
    }

    if (dwReason == DLL_PROCESS_ATTACH) {
        DetourRestoreAfterWith();

        DetourTransactionBegin();
        DetourUpdateThread(GetCurrentThread());
        DetourAttach(&(PVOID&)TrueSleep, TimedSleep);
        DetourTransactionCommit();
    }
    else if (dwReason == DLL_PROCESS_DETACH) {
        DetourTransactionBegin();
        DetourUpdateThread(GetCurrentThread());
        DetourDetach(&(PVOID&)TrueSleep, TimedSleep);
        DetourTransactionCommit();
    }
    return TRUE;
}
```

Figure 5. Simple detour to modify the Windows Sleep API.

Interception of the target function is enabled by invoking the DetourAttach API within a detour transaction. A detour transaction is marked by calls to the DetourTransactionBegin API and the DetourTransactionCommit API. The DetourAttach API takes two arguments: the address of the target pointer and the pointer to the detour function. The target function is not given as an argument because it must already be stored in the target pointer.

The DetourUpdateThread API enlists threads in the transaction so that their instruction pointers are appropriately updated when the transaction commits.

The DetourAttach API allocates and prepares a trampoline for calling the target function. When the detour transaction commits, the target function and trampoline are rewritten and the target pointer is updated to point to the trampoline function.

Once a target function has been detoured, any call to the target function will be re-routed through the detour function. 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 becomes simply a subroutine callable by the detour function.

Interception of a target function is removed by calling the DetourDetach API within a detour transaction. Like the DetourAttach API, the DetourDetach API takes two arguments: the address of the target pointer and the pointer to the detour function. When the detour transaction commits, the target function is rewritten and restored to its original code, the trampoline function is deleted, and the target pointer is restored to point to the original target function.

In cases where detour functions need to inserted into an existing application without source code access, the detour functions should be packaged in a DLL. The DLL can be loaded into a new process at creation time using the DetourCreateProcessWithDllEx or DetourCreateProcessWithDlls APIs. If a DLL is inserted using `DetourCreateProcessWithDllEx` or `DetourCreateProcessWithDlls`, the DllMain function must call the DetourRestoreAfterWith API. If the DLL may be used in mixed 32-bit and 64-bit environments, then the DllMain function must call the DetourIsHelperProcess API. The DLL must export the DetourFinishHelperProcess API as export Ordinal 1, which will be called by `rundll32.exe` to perform the helper tasks.

NOTE: Microsoft in no way warrants or supports any Microsoft or third-party code that has been altered either with a detour or with any other mechanism.

The withdll.exe program include in the Detours package uses the DetourCreateProcessWithDlls API to start a new process with a named DLL.

Payloads and DLL Import Editing

In addition to APIs for attaching and detaching detours functions, the Detours package also include APIs for attaching arbitrary data segments, called *payloads*, to Windows binary files and for editing DLL import tables. The binary editing APIs in Detours are fully reversible; Detours stores recovery information within the binary to enable removal of the edits at any time in the future.

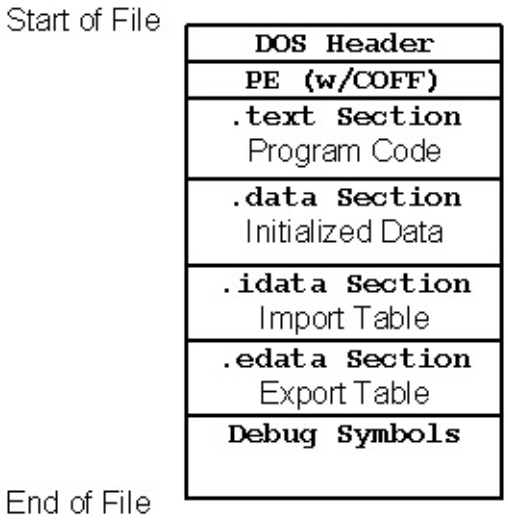

Figure 3. Format of a Windows PE binary file.

Figure 3 shows the basic structure of a Windows Portable Executable (PE) binary file. The PE format for Windows binaries is an extension of COFF (the Common Object File Format). A Windows 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.

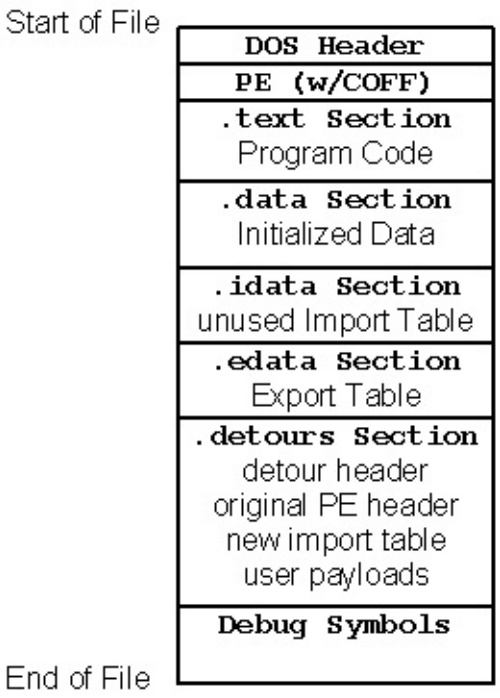

Figure 4. Format of a Detours-modified binary file.

To modify a Windows binary, Detours creates a new `.detours` section between the export table and the debug symbols, as shown in Figure 4. Note that debug symbols must always reside last in a Windows 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 Windows 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 Windows 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 Windows file. Second, the new import table can contain renamed import DLLs and functions or entirely new DLLs and functions. For example, the setdll.exe program included in the Detours package, inserts an initial entry for a user's DLL into a target application binary. As the first entry in the application's import

table, the user's DLL is always the first DLL to run in the application's address space.

Detours provides APIs for editing import tables (DetourBinaryEditImports), adding payloads (DetourBinarySetPayload), enumerating payloads (DetourBinaryEnumeratePayloads), and removing payloads (DetourBinaryPurgePayloads). Detours also provides APIs for enumerating the binary files mapped into an address space (DetourEnumerateModules) and locating payloads within those mapped binaries (DetourFindPayload). Each payload is identified by a 128-bit globally unique identifier (GUID). Payloads can be used to attach per-application configuration data to application binaries.

Payloads can be copied directly into a target process using the DetourCopyPayloadToProcess API.

Detouring 32-bit and 64-bit Processes

Note: Only the Professional editions of Detours support 64-bit processes. The non-commercial, Express editions of Detours only support 32-bit x86 processes.

The most common usage scenario for Detours is to detour functions in an existing application without modifying the orginal application binaries. In these scenarios, the user-supplied detour functions are packaged in a DLL that is loaded into the applicaiton at startup time using the DetourCreateProcessWithDll API. The DetourCreateProcessWithDll API is called from the *parent process*; it alters the in-memory copy of the application by inserting an import table entry for the detour DLL. This new import table entry causes the OS loader to load the DLL after the application process has started, but before any of the application code can run. The detour DLL then has a chance to hook target functions in the *target process*.

In computers with 64-bit processors, Windows supports both 32-bit and 64-bit applications. To support both 32-bit and 64-bit applications, you must create both 32-bit and 64-bit versions of your detour DLL. You must also replace all uses of the DetourCreateProcessWithDll API with either the DetourCreateProcessWithDllEx API or DetourCreateProcessWithDlls API. The DetourCreateProcessWithDllEx and DetourCreateProcessWithDlls APIs chooses between the 32-bit or 64-bit versions of your DLL as appropriate for the target application.

What To Do

To support both 32-bit and 64-bit applications on a single system, you must create two DLLs. One DLL must contain 32-bit code, the other DLL must contain 64-bit code. The DLLs must reside in the same directory and must have identical names except that the name of the 32-bit DLL should end with "32" and the name of the 64-bit DLL should end with "64". For example, matching DLLs would be named `foo32.dll` and `foo64.dll`.

You should use the DetourCreateProcessWithDllEx or DetourCreateProcessWithDlls API to start a process with your DLL. Furthermore, your DLLs must:

Export the DetourFinishHelperProcess API as export ordinal 1.

Call the DetourIsHelperProcess API in your `DllMain` function. Immediately return TRUE if DetourIsHelperProcess return TRUE.

Call the DetourCreateProcessWithDllEx or DetourCreateProcessWithDlls API instead of DetourCreateProcessWithDll to create new target processes.

How It Works

In the case where both the parent process and the target process are the same, such as both 32-bit or both 64-bit, the DetourCreateProcessWithDllEx API works the same as the DetourCreateProcessWithDll API.

When the parent process is 32-bit and the target is 64-bit or the parent is 64-bit and the target is 32-bit, DetourCreateProcessWithDllEx creates a *helper process* by loading your DLL into a `rundll32.exe` process, and calling DetourFinishHelperProcess through export Ordinal 1. This API patches up the application's import table using the correct 32-bit or 64-bit code.

Give It a Try

To give helper processes a try, first build the Detours samples for 32-bit using a 32-bit build environment. Then build the samples for 64-bit using a 64-bit build environment. Then in the `samples\tryman` directory, in the 64-bit environment type "`nmake size64`" to run a recursive set of processes that alternate between 32-bit and 64-bit processes.

Remarks

For more information on `rundll32.exe`, see http://support.microsoft.com/kb/164787.

Related APIs:

DetourCreateProcessWithDllEx, DetourCreateProcessWithDlls, DetourFinishHelperProcess, DetourIsHelperProcess, DetourRestoreAfterWith.

Related Samples

Simple, Simple, Slept, Traceapi, Tracebld, Tracelnk, Tracemem, Tracereg, Traceser, Tracetcp, Tryman.

Detours API Reference

The Table of Contents provides an alphabetical listing of the available API functions, which can be grouped as follows:

APIs For Detouring Target Functions

- DetourTransactionBegin
- DetourUpdateThread
- DetourAttach
- DetourAttachEx
- DetourDetach
- DetourSetIgnoreTooSmall
- DetourSetRetainRegions
- DetourSetSystemRegionLowerBound
- DetourSetSystemRegionUpperBound
- DetourTransactionAbort
- DetourTransactionCommit
- DetourTransactionCommitEx

APIs For Finding Target Functions

- DetourFindFunction
- DetourCodeFromPointer

APIs For Accessing Loaded Binaries and Payloads

- DetourEnumerateModules
- DetourGetEntryPoint
- DetourGetModuleSize
- DetourEnumerateExports
- DetourEnumerateImport
- DetourEnumerateImportEx
- DetourFindPayload
- DetourGetContainingModule
- DetourGetSizeOfPayloads

APIs For Modifying Binaries

- DetourBinaryOpen
- DetourBinaryEnumeratePayloads
- DetourBinaryFindPayload
- DetourBinarySetPayload
- DetourBinaryDeletePayload
- DetourBinaryPurgePayloads
- DetourBinaryEditImports
- DetourBinaryResetImports
- DetourBinaryWrite
- DetourBinaryClose

APIs For Inserting DLLs and Payloads Into New Processes

- DetourCreateProcessWithDllEx
- DetourCreateProcessWithDlls
- DetourCopyPayloadToProcess
- DetourFinishHelperProcess
- DetourIsHelperProcess
- DetourRestoreAfterWith

DetourAttach

Attach a detour to a target function.

Definition

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

Parameters

ppPointer
Pointer to the target pointer to which the detour will be attached.
pDetour
Pointer to the detour function.

Return value

Returns NO_ERROR if successful; otherwise, returns an error code.

Error codes

ERROR_INVALID_BLOCK
The function referenced is too small to be detoured.
ERROR_INVALID_HANDLE
The ppPointer parameter is null or points to a null pointer.
ERROR_INVALID_OPERATION
No pending transaction exists.
ERROR_NOT_ENOUGH_MEMORY
Not enough memory exists to complete the operation.

Remarks

`DetourAttach` attaches a detour to a target function as part of the current transaction opened by the DetourTransactionBegin API.

For more information on using Detours to intercept function calls, see Interception of Binary Functions or Using Detours in the Detours Overview.

Related Samples

Commem, Cping, Dtest, Excep, FindFunc, Simple, Slept, Traceapi, Tracebld, Tracelnk, Tracemem, Tracereg, Traceser, Tracetcp, Tryman.

DetourAttachEx

Attach a detour to a target function and retrieve additional detail about the ultimate target.

Definition

```
LONG DetourAttachEx(
    _Inout_   PVOID * ppPointer,
    _In_      PVOID pDetour,
    _Out_opt_ PDETOUR_TRAMPOLINE * ppRealTrampoline
    _Out_opt_ PVOID * ppRealTarget
    _Out_opt_ PVOID * ppRealDetour
    );
```

Parameters

ppPointer
: Pointer to the target pointer to which the detour will be attached.

pDetour
: Pointer to the detour function.

ppRealTrampoline
: Variable to receive the address of the trampoline.

ppRealTarget
: Variable to receive the final address of the target function.

ppRealDetour
: Variable to receive the final address of the detour function.

Return value

Returns NO_ERROR if successful; otherwise, returns an error code.

Error codes

ERROR_INVALID_BLOCK
: The function referenced is too small to be detoured.

ERROR_INVALID_HANDLE
: The ppPointer parameter is null or points to a null pointer.

ERROR_INVALID_OPERATION
No pending transaction exists.
ERROR_NOT_ENOUGH_MEMORY
Not enough memory exists to complete the operation.

Remarks

`DetourAttachEx` attaches a detour to a target function and retrieves additional detail about the ultimate target as part of the current transaction opened by the DetourTransactionBegin API.

For more information on using Detours to intercept function calls, see Interception of Binary Functions or Using Detours in the Detours Overview.

DetourBinaryClose

Close a binary opened for editing.

Definition

```
BOOL DetourBinaryClose(
    _In_ PDETOUR_BINARY pBinary
    );
```

Parameters

pBinary
Pointer to the binary opened by DetourBinaryOpen, .

Return value

If successful, returns `TRUE`; otherwise, returns `FALSE`.

Remarks

`DetourBinaryClose` closes the binary opened for editing by DetourBinaryOpen.

For more information on binary editing with Detours, see Payloads and DLL Import Editing in the Detours Overview.

Related Samples

Dumpi, Impmunge, Setdll.

DetourBinaryDeletePayload

Remove a payload from a binary.

Definition

```
BOOL DetourBinaryDeletePayload(
    _In_ PDETOUR_BINARY pBinary,
    _In_ REFGUID rguid
    );
```

Parameters

pBinary
Pointer to the binary opened by DetourBinaryOpen, .
rguid
GUID of payload to remove.

Remarks

`DetourBinaryDeletePayload` removes a payload from a binary opened by DetourBinaryOpen .

For more information on binary editing with Detours and paylods, see Payloads and DLL Import Editing in the Detours Overview.

Return value

If successful, returns `TRUE`; otherwise, returns `FALSE`.

DetourBinaryEditImports

Edit the import tables of a binary.

Definition

```
BOOL DetourBinaryEditImports(
    _In_     PDETOUR_BINARY pBinary,
    _In_opt_ PVOID pContext,
    _In_opt_ PF_DETOUR_BINARY_BYWAY_CALLBACK pfByway,
    _In_opt_ PF_DETOUR_BINARY_FILE_CALLBACK pfFile,
    _In_opt_ PF_DETOUR_BINARY_SYMBOL_CALLBACK pfSymbol,
    _In_opt_ PF_DETOUR_BINARY_COMMIT_CALLBACK pfFinal
    );
```

Parameters

pBinary
: Pointer to binary opened by `DetourBinaryOpen` .

pContext
: Program specific context pointer to be passed unmodified to each callback function.

pfByway
: Callback function called before each module in the import table.

pfFile
: Callback function called once for each module in the import table.

pfSymbol
: Callback function called once for each symbol in the import table.

pfCommit
: Callback function called at the end of the import table if there have been no errors.

Return value

If successful, returns `TRUE`; otherwise, returns `FALSE`.

Remarks

`DetourBinaryEditImports` edits the import tables of a binary opened by `DetourBinaryOpen`. Detours stores edits in a reversible format using a Detours payload. The `DetourBinaryResetImports` function can be used to remove the edits.

`DetourBinaryEditImports` walks sequentially through the import table of a binary making callbacks on all points of interest. Four points of interest are currently supported, each with its own callback function:

- **Files**: The `pfFile` function is called for each file listed in the import table. The callback function can alter the file name at its discretion.
- **Symbols**: The `pfSymbol` function is called for each symbol listed in each file in the import table. The callback function can alter the symbol name at its discretion.
- **Byways**: The `pfByway` function is called once at the start of the import table, between each pair of import functions, and again at the end of the import table. The `pfByway` function can at its discretion introduce a new import file into the import table. When a byway is inserted, the import table is modified to import the function exported with ordinal #1 from the named import file.
- **Commit**: The `pfCommit` function is called at the end of walking the import table if no errors have been returned by previous callback functions.

Consider a binary that imports the functions `CreateFileW` and `CloseHandle` from `Kernel32.dll`, the function `CommandLineToArgvW` from `Shell32.dll`, the functions `RegOpenKeyExW`, `RegQueryValueW`, and `RegCloseKey` from `AdvApi32.dll`, and has a byway for `MyDetour.Dll` inserted from a previous call to `DetoursBinaryEditImports`. A program calling `DetoursBinaryEditImports` would receive the following callbacks:

- `BywayCallback (..., NULL, ...)`
- `BywayCallback (..., "MyDetour.dll", ... )`
- `BywayCallback (..., NULL, ...)`
- `FileCallback (..., "Kernel32.dll", ...)`
- `SymbolCallback (..., "CloseHandle", ...)`
- `SymbolCallback (..., "CreateFileW", ...)`
- `BywayCallback (..., NULL, ...)`
- `FileCallback (..., "Shell32.dll", ...)`
- `SymbolCallback (..., "CommandLineToArgvW", ...)`

- `BywayCallback (..., NULL, ...)`
- `FileCallback (..., "AdvApi32.dll", ...)`
- `SymbolCallback (..., "RegCloseKey", ...)`
- `SymbolCallback (..., "RegQueryValueW", ...)`
- `SymbolCallback (..., "RegOpenKeyExW", ...)`
- `BywayCallback (..., NULL, ...)`
- `CommitCallback (...)`

For more information on binary editing with Detours, see Payloads and DLL Import Editing in the Detours Overview.

Note: Each DLL inserted as a byway must export a function with ordinal #1. If the export table for the DLL does not export a function with ordinal #1, the target binary will fail to load correct.

Related Samples

Dumpi, Impmunge, Setdll.

PF_DETOUR_BINARY_BYWAY_CALLBACK

Pointer to function called once for each existing byway or opportunity to insert a new byway while editing an import table using the DetourBinaryEditImports API.

Definition

```
BOOL BinaryBywayCallback(
    _In_opt_                  PVOID pContext,
    _In_opt_                  LPCSTR pszFile,
    _Outptr_result_maybenull_ LPCSTR * ppszOutFile
    );
```

Parameters

pContext
: Umodified program specific context pointer passed as pContext argument to DetourBinaryEditImports.

pszFile
: Name of byway listed in current import table or NULL.

ppszOutFile
: Pointer to output name of desired byway.

Return value

`TRUE` to continue editing import table or `FALSE` to abort.

Remarks

`PF_DETOUR_BINARY_BYWAY_CALLBACK` is called once for each existing byway in the target binary and once before or after each file or byway listed in the existing import table. When called for an existing byway, `pszFile` will have a non-NULL value.

When `PF_DETOUR_BINARY_BYWAY_CALLBACK` is called before or after an existing file or byway, `pszFile` will be NULL. The callback function can use this

opportunity to insert a new byway if desired.

Note: Each DLL inserted as a byway must export a function with ordinal #1. If the export table for the DLL does not export a function with ordinal #1, the target binary will fail to load correct.

PF_DETOUR_BINARY_COMMIT_CALLBACK

Pointer to function called at the end of editing an import table using the DetourBinaryEditImports API.

Definition

```
BOOL BinaryCommitCallback(
    _In_opt_ PVOID pContext
    );
```

Parameters

pContext
: Umodified program specific context pointer passed as pContext argument to DetourBinaryEditImports.

Return value

`TRUE` to continue editing import table or `FALSE` to abort.

PF_DETOUR_BINARY_FILE_CALLBACK

Pointer to function called once for file while editing an import table using the DetourBinaryEditImports API.

Definition

```
BOOL BinaryFileCallback(
    _In_opt_                  PVOID pContext,
    _In_                      LPCSTR pszOrigFile,
    _In_                      LPCSTR pszFile,
    _Outptr_result_maybenull_ LPCSTR * ppszOutFile
    );
```

Parameters

pContext
: Umodified program specific context pointer passed as pContext argument to DetourBinaryEditImports.

pszOrigFile
: Name of file listed in original import table.

pszFile
: Name of file listed in current import table

ppszOutFile
: Pointer to output desired import table name.

Return value

`TRUE` to continue editing import table or `FALSE` to abort.

PF_DETOUR_BINARY_SYMBOL_CALLBACK

Pointer to function called once for each symbol while editing an import table using the DetourBinaryEditImports API.

Definition

```
BOOL BinarySymbolCallback(
    _In_opt_                  PVOID pContext,
    _In_                      ULONG nOrigOrdinal,
    _In_                      ULONG nOrdinal,
    _Out_                     ULONG * pnOutOrdinal,
    _In_opt_                  PCSTR pszOrigSymbol,
    _In_opt_                  PCSTR pszSymbol,
    _Outptr_result_maybenull_ PCSTR *ppszOutSymbol
    );
```

Parameters

pContext
: Umodified program specific context pointer passed as pContext argument to DetourBinaryEditImports.

nOrigOrdinal
: Import ordinal listed in original import table.

nOrdinal
: Import ordinal listed in current import table.

pnOutOrdinal
: Pointer to output desired import ordinal.

pszOrigSymbol
: Import symbol listed in original import table.

pszSymbol
: Import symbol listed in current import table.

ppszOutSymbol
: Pointer to output desire import symbol.

Return value

`TRUE` to continue editing import table or `FALSE` to abort.

DetourBinaryEnumeratePayloads

Enumerate the payloads in a binary.

Definition

```
_Writable_bytes_(*pcbData)
_Readable_bytes_(*pcbData)
_Success_(return != NULL)
PVOID DetourBinaryEnumeratePayloads(
    _In_      PDETOUR_BINARY pBinary,
    _Out_opt_ GUID * pGuid,
    _Out_     DWORD * pcbData,
    _Inout_   DWORD * pnIterator
    );
```

Parameters

pBinary
: Pointer to the binary opened by DetourBinaryOpen .

pGuid
: Pointer to the variable to receive the GUID of the next payload.

pcbData
: Poiunter to the Variable to receive the size in bytes of the next payload.

pnIterator
: Enumeration variable. Should be set to zero to start enumeration. The enumeration variable should not be modified between successive calls to this function.

Remarks

`DetourBinaryEnumeratePayloads` enumerates all of the payloads in a binary opened by DetourBinaryOpen .

For more information on binary editing with Detours and paylods, see Payloads and DLL Import Editing in the Detours Overview.

Return value

If successful, returns a pointer to the next payload; otherwise, returns `NULL`.

DetourBinaryFindPayload

Find a payload within a binary.

Definition

```
_Writable_bytes_(*pcbData)
_Readable_bytes_(*pcbData)
_Success_(return != NULL)
PVOID DetourBinaryFindPayload(
    _In_  PDETOUR_BINARY pBinary,
    _In_  REFGUID rguid,
    _Out_ DWORD * pcbData
    );
```

Parameters

pBinary
: Pointer to the binary opened by DetourBinaryOpen .

rguid
: GUID of the specified payload.

pcbData
: Pointer to the variable that receives the size of the specified payload in bytes.

Remarks

`DetourBinaryFindPayload`Finds a specific payload within a binary opened by DetourBinaryOpen.

For more information on binary editing with Detours and paylods, see Payloads and DLL Import Editing in the Detours Overview.

Return value

If successful, returns `TRUE`; otherwise, returns `FALSE`.

DetourBinaryOpen

Read the contents of a binary into memory for editing.

Definition

```
PDETOUR_BINARY DetourBinaryOpen(
    _In_ HANDLE hFile
    );
```

Parameters

hFile
: The handle of the binary to be opened for editing.

Return value

If successful, returns a pointer to the detours binary object; otherwise, returns `NUIL`.

Error codes

The function sets the following error code, if appropriate. The error code may be retrived after the function has returned by calling `GetLastError`.

ERROR_OUT_OF_MEMORY
: Not enough memory to complete the operation.

Remarks

`DetourBinaryOpen` reads the contents of a Windows PE COFF binary into memory for editing.

For more information on binary editing with Detours, see Payloads and DLL Import Editing in the Detours Overview.

Related Samples

Dumpi, Impmunge, Setdll.

DetourBinaryPurgePayloads

Remove all payloads from a binary.

Definition

```
BOOL DetourBinaryPurgePayloads(
    _In_ PDETOUR_BINARY pBinary
    );
```

Parameters

pBinary
Pointer to binary, opened by DetourBinaryOpen, to be purged.

Remarks

`DetourBinaryPurgePayloads`Removes all payloads from a binary opened by DetourBinaryOpen.

For more information on binary editing with Detours, see Payloads and DLL Import Editing in the Detours Overview.

Return value

If successful, returns `TRUE`; otherwise, returns `FALSE`.

DetourBinaryResetImports

Remove all edits by Detours of the binary's import table.

Definition

```
BOOL DetourBinaryResetImports(
    _In_ PDETOUR_BINARY pBinary
    );
```

Parameters

pBinary
Pointer to the binary opened by DetourBinaryOpen.

Return value

If successful, returns `TRUE`; otherwise, returns `FALSE`.

Remarks

`DetourBinaryResetImports`Removes all Detours edits made to the import table of a binary opened by DetourBinaryOpen.

For more information on binary editing with Detours, see Payloads and DLL Import Editing in the Detours Overview.

Related Samples

Setdll.

DetourBinarySetPayload

Attach a payload to a binary.

Definition

```
PVOID DetourBinarySetPayload(
    _In_                    PDETOUR_BINARY pBinary,
    _In_                    REFGUID rguid,
    _In_reads_opt_(cbData) PVOID pData,
    _In_                    DWORD cbData
    );
```

Parameters

pBinary
: Pointer to the binary opened by DetourBinaryOpen

rguid
: GUID of the payload to be added to binary.

pData
: Pointer to payload data.

cbData
: Size of the payload data in bytes.

Remarks

`DetourBinarySetPayload` attaches a payload to a binary opened by DetourBinaryOpen. The payload can be located at runtime using the DetourFindPayload API.

For more information on binary editing with Detours, see Payloads and DLL Import Editing in the Detours Overview.

Return value

If successful, returns `TRUE`; otherwise, returns `FALSE`.

DetourBinaryWrite

Write an updated binary to a file.

Definition

```
BOOL DetourBinaryWrite(
    _In_ PDETOUR_BINARY pBinary,
    _In_ HANDLE hFile
    );
```

Parameters

pBinary
: Pointer to the binary to be written to a file.

hFile
: Handle of the file to receive the contents of the binary.

Return value

If successful, returns `TRUE`; otherwise, returns `FALSE`.

Remarks

`DetourBinaryWrite` writes the updated binary, opened by DetourBinaryOpen, to a file.

For more information on binary editing with Detours, see Payloads and DLL Import Editing in the Detours Overview.

Related Samples

Impmunge, Setdll.

DetourCodeFromPointer

Return a pointer to the code that implements a function pointer.

Definition

```
PVOID DetourCodeFromPointer(
    _In_      PVOID pPointer,
    _Out_opt_ PVOID *ppGlobals
    );
```

Parameters

pPointer
: Pointer to the function.

ppGlobals
: Variable to receive the address of the function's globals data.

Return value

Returns a pointer to the code implementing the function.

Remarks

`DetourCodeFromPointer` returns a pointer to the code that implements a function pointed to by a function pointer. When a binary is statically linked against a DLL, pointers to functions from the DLL often point not to the code from the DLL, but to a jump statement in the binary's import table. `DetourCodeFromPointer` returns the address of the actual target function, not the jump statement.

On some platforms, function pointer point to indirection labels, not code. On these platforms, the indirection labels also contains a pointer to the global (or static) data associated with the function.

For more information on using Detours to intercept function calls, see Interception of Binary Functions or Using Detours in the Detours Overview.

Related Samples

Slept, Tracebld.

DetourCopyPayloadToProcess

Copy a payload into a target process.

Definition

```
BOOL DetourCopyPayloadToProcess(
    _In_                      HANDLE hProcess,
    _In_                      REFGUID rguid,
    _In_reads_bytes_(cbData) PVOID pvData,
    _In_                      DWORD cbData
    );
```

Parameters

hProcess
: Process into which payload should be copied.

rguid
: GUID of the specified payload.

pvData
: Pointer to payload data.

pcbData
: Size in bytes of payload data.

Return value

Returns TRUE if the payload was successfully copied to the target process.

Error codes

On failure, `DetourCopyPayloadToProcess` will return FALSE. Extended error code information may be retrieved by calling `GetLastError`.

Remarks

`DetourCopyPayloadToProcess` allocated a region of memory in the target process using the `VirtualAllocEx` API. It then uses the WriteProcessMemory

API to create an artificial PE binary module in the target memory. In the artificial module, `DetourCopyPayloadToProcess` creates a `.detours` section with the specified payload data.

Code in the target process can find the payload by enumerating through all modules using the DetourEnumerateModules API and querying each module for the payload using the DetourFindPayload API.

Related Samples

Tracebld, WithDll.

DetourCreateProcessWithDll

Create a new process and load a DLL into it..

DetourCreateProcessWithDll has been deprecated. Use DetourCreateProcessWithDllEx or DetourCreateProcessWithDlls instead.

Definition

```
BOOL DetourCreateProcessWithDll(
    _In_opt_    LPCTSTR lpApplicationName,
    _Inout_opt_ LPTSTR lpCommandLine,
    _In_opt_    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    _In_opt_    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    _In_        BOOL bInheritHandles,
    _In_        DWORD dwCreationFlags,
    _In_opt_    LPVOID lpEnvironment,
    _In_opt_    LPCTSTR lpCurrentDirectory,
    _In_        LPSTARTUPINFOW lpStartupInfo,
    _Out_       LPPROCESS_INFORMATION lpProcessInformation,
    _In_        LPCSTR lpDllName,
    _In_opt_    PDETOUR_CREATE_PROCESS_ROUTINEW pfCreateProcessW
    );
```

Parameters

lpApplicationName
: Application name as defined for CreateProcess API.

lpCommandLine
: Command line as defined for CreateProcess API.

lpProcessAttributes
: Process attributes as defined for CreateProcess API.

lpThreadAttributes
: Thread attributes as defined for CreateProcess API.

bInheritHandles
: Inherit handle flags as defined for CreateProcess API.

dwCreationFlags
: Creation flags as defined for CreateProcess API.

lpEnvironment
: Process environment variables as defined for CreateProcess API.

lpCurrentDirectory
: Process current directory as defined for CreateProcess API.

lpStartupInfo
: Process startup information as defined for CreateProcess API.

lpProcessInformation
: Process handle information as defined for CreateProcess API.

lpDllName
: Pathname of the DLL to be insert into the new process.

pfCreateProcessW
: Pointer to program specific replacement for the CreateProcess API, or NULL if the standard CreateProcess API should be used to create the new process.

Return value

Returns `TRUE` if the new process was created; otherwise returns `FALSE`.

Error codes

See error code returned from CreateProcess.

Remarks

`DetourCreateProcessWithDll` creates a new process with the specified DLL inserted into it.

The process is created in a suspended state with the CREATE_SUSPENDED flag to CreateProcess. Detours then modifies the image of the application binary in the new process to include the specified DLL as its first import. Execution in the process is then resumed. When execution resumes, the Windows process loader will first load the detour DLL and then any other DLLs in the application's import table, before calling the application entry point.

`DetourCreateProcessWithDll` modifies the in-memory import table of the target PE binary program in the new process it creates. The updated import table will contain a reference to function ordinal #1 exported from the detour DLL.

Note: The new process will fail to start if the detour DLL does not export

DetourFinishHelperProcess as export ordinal #1.

After the detour DLL has been loaded, it should reverse changes to the in-memory import table by calling `DetourRestoreAfterWith`. To facilitate reversing these changes, `DetourCreateProcessWithDll` copies relevant reversal data into a payload in the target process using the `DetourCopyPayloadToProcess` API. The loaded DLL should call the `DetourRestoreAfterWith` API to restores the contents of the import table.

Related Samples

Tracebld, Tracemem, Tracereg, Traceser, Withdll.

DetourCreateProcessWithDllEx

Create a new process and load a DLL into it. Chooses the appropriate 32-bit or 64-bit DLL based on the target process.

Replaces DetourCreateProcessWithDll.

Definition

```
BOOL DetourCreateProcessWithDllEx(
    _In_opt_    LPCTSTR lpApplicationName,
    _Inout_opt_ LPTSTR lpCommandLine,
    _In_opt_    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    _In_opt_    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    _In_        BOOL bInheritHandles,
    _In_        DWORD dwCreationFlags,
    _In_opt_    LPVOID lpEnvironment,
    _In_opt_    LPCTSTR lpCurrentDirectory,
    _In_        LPSTARTUPINFOW lpStartupInfo,
    _Out_       LPPROCESS_INFORMATION lpProcessInformation,
    _In_        LPCSTR lpDllName,
    _In_opt_    PDETOUR_CREATE_PROCESS_ROUTINEW pfCreateProcessW
    );
```

Parameters

lpApplicationName
: Application name as defined for CreateProcess API.

lpCommandLine
: Command line as defined for CreateProcess API.

lpProcessAttributes
: Process attributes as defined for CreateProcess API.

lpThreadAttributes
: Thread attributes as defined for CreateProcess API.

bInheritHandles
: Inherit handle flags as defined for CreateProcess API.

dwCreationFlags
: Creation flags as defined for CreateProcess API.

lpEnvironment
: Process environment variables as defined for CreateProcess API.

lpCurrentDirectory
: Process current directory as defined for CreateProcess API.

lpStartupInfo
: Process startup information as defined for CreateProcess API.

lpProcessInformation
: Process handle information as defined for CreateProcess API.

lpDllName
: Pathname of the DLL to be insert into the new process. To support both 32-bit and 64-bit applications, The DLL name should end with "32" if the DLL contains 32-bit code and should end with "64" if the DLL contains 64-bit code. If the target process differs in size from the parent process, Detours will automatically replace "32" with "64" or "64" with "32" in the path name.

pfCreateProcessW
: Pointer to program specific replacement for the CreateProcess API, or NULL if the standard CreateProcess API should be used to create the new process.

Return value

Returns `TRUE` if the new process was created; otherwise returns `FALSE`.

Error codes

See error code returned from CreateProcess.

Remarks

`DetourCreateProcessWithDllEx` creates a new process with the specified DLL inserted into it.

The process is created in a suspended state with the CREATE_SUSPENDED flag to CreateProcess. Detours then modifies the image of the application binary in the new process to include the specified DLL as its first import. Execution in the process is then resumed. When execution resumes, the Windows process loader will first load the target DLL and then any other DLLs in the application's import table, before calling the application entry point.

`DetourCreateProcessWithDllEx` modifies the in-memory import table of the target PE binary program in the new process it creates. The updated import table will contain a reference to function ordinal #1 exported from the target DLL. If the target process is 32-bit and the parent process is 64-bit, or if the target process is 64-bit and the parent process is 32-bit, `DetourCreateProcessWithDllEx` will use `rundll32.exe` to load the DLL into a helper process that matches the target process temporarily in order to update the target processes import table with the correct DLL.

Note: The new process will fail to start if the target DLL does not contain a exported function with ordinal #1.

After the target DLL has been loaded, it can reverse changes to the in-memory import table by calling `DetourRestoreAfterWith`. To facilitate reversing these changes, `DetourCreateProcessWithDllEx` copies relevant reversal data into a payload in the target process using the `DetourCopyPayloadToProcess` API. The loaded DLL should call the `DetourRestoreAfterWith` API to restores the contents of the import table.

Related Samples

Traceapi, Tracebld, Tracemem, Tracereg, Traceser, Tryman, Withdll.

DetourCreateProcessWithDlls

Create a new process and load DLLs into it. Chooses the appropriate 32-bit or 64-bit DLL based on the target process.

Replaces DetourCreateProcessWithDll.

Definition

```
BOOL DetourCreateProcessWithDlls(
    _In_opt_          LPCTSTR lpApplicationName,
    _Inout_opt_       LPTSTR lpCommandLine,
    _In_opt_          LPSECURITY_ATTRIBUTES lpProcessAttributes,
    _In_opt_          LPSECURITY_ATTRIBUTES lpThreadAttributes,
    _In_              BOOL bInheritHandles,
    _In_              DWORD dwCreationFlags,
    _In_opt_          LPVOID lpEnvironment,
    _In_opt_          LPCTSTR lpCurrentDirectory,
    _In_              LPSTARTUPINFOW lpStartupInfo,
    _Out_             LPPROCESS_INFORMATION lpProcessInformation,
    _In_              DWORD nDlls,
    _In_reads_(nDlls) LPCSTR *rlpDlls,
    _In_opt_          PDETOUR_CREATE_PROCESS_ROUTINEW pfCreateProc
    );
```

Parameters

lpApplicationName
: Application name as defined for CreateProcess API.

lpCommandLine
: Command line as defined for CreateProcess API.

lpProcessAttributes
: Process attributes as defined for CreateProcess API.

lpThreadAttributes
: Thread attributes as defined for CreateProcess API.

bInheritHandles
: Inherit handle flags as defined for CreateProcess API.

dwCreationFlags
: Creation flags as defined for CreateProcess API.

lpEnvironment

Process environment variables as defined for CreateProcess API.

lpCurrentDirectory

Process current directory as defined for CreateProcess API.

lpStartupInfo

Process startup information as defined for CreateProcess API.

lpProcessInformation

Process handle information as defined for CreateProcess API.

nDlls

Count of the number of DLLs in *rlpDlls*.

rlpDlls

Array of pathnames of the DLL to be insert into the new process. To support both 32-bit and 64-bit applications, The DLL names should end with "32" if the DLL contains 32-bit code and should end with "64" if the DLL contains 64-bit code. If the target process differs in size from the parent process, Detours will automatically replace "32" with "64" or "64" with "32" in the path name.

pfCreateProcessW

Pointer to program specific replacement for the CreateProcess API, or NULL if the standard CreateProcess API should be used to create the new process.

Return value

Returns `TRUE` if the new process was created; otherwise returns `FALSE`.

Error codes

See error code returned from CreateProcess.

Remarks

`DetourCreateProcessWithDlls` creates a new process with the specified DLL inserted into it.

The process is created in a suspended state with the CREATE_SUSPENDED flag to CreateProcess. Detours then modifies the image of the application binary in the new process to include the specified DLL as its first import. Execution in the process is then resumed. When execution resumes, the Windows process

loader will first load the target DLL and then any other DLLs in the application's import table, before calling the application entry point.

`DetourCreateProcessWithDlls` modifies the in-memory import table of the target PE binary program in the new process it creates. The updated import table will contain a reference to function ordinal #1 exported from the target DLL. If the target process is 32-bit and the parent process is 64-bit, or if the target process is 64-bit and the parent process is 32-bit, `DetourCreateProcessWithDlls` will use `rundll32.exe` to load the DLL into a helper process that matches the target process temporarily in order to update the target processes import table with the correct DLL.

Note: The new process will fail to start if the target DLL does not contain a exported function with ordinal #1.

After the target DLL has been loaded, it can reverse changes to the in-memory import table by calling `DetourRestoreAfterWith`. To facilitate reversing these changes, `DetourCreateProcessWithDlls` copies relevant reversal data into a payload in the target process using the `DetourCopyPayloadToProcess` API. The loaded DLL should call the `DetourRestoreAfterWith` API to restores the contents of the import table.

Related Samples

Traceapi, Tracebld, Tracemem, Tracereg, Traceser, Tryman, Withdll.

DetourDetach

Detach a detour from a target function.

Definition

```
LONG DetourDetach(
    _Inout_ PVOID * ppPointer,
    _In_    PVOID pDetour
    );
```

Parameters

ppPointer
Pointer to the target pointer from which the detour will be detached.
pDetour
Pointer to the detour function.

Return value

Returns NO_ERROR if successful; otherwise, returns an error code.

Error codes

ERROR_INVALID_BLOCK
The function to be detached was too small to be detoured.
ERROR_INVALID_HANDLE
The ppPointer parameter is null or references a null address.
ERROR_INVALID_OPERATION
No pending transaction exists.
ERROR_NOT_ENOUGH_MEMORY
Not enough memory to complete the operation.

Remarks

`DetourDetach` detaches a detour from a target function as part of the current transaction opened by the DetourTransactionBegin API.

For more information on using Detours to intercept function calls, see Interception of Binary Functions or Using Detours in the Detours Overview.

Related Samples

Commem, Cping, FindFunc, Member, Simple, Slept, Traceapi, Tracebld, Tracelnk, Tracemem, Tracereg, Traceser, Tracetcp, Tryman.

DetourEnumerateExports

Enumerate exports from a module.

Definition

```
BOOL DetourEnumerateExports(
    _In_     HMODULE hModule,
    _In_opt_ PVOID pContext,
    _In_     PF_DETOUR_ENUMERATE_EXPORT_CALLBACK pfExport
    );
```

Parameters

hModule
: The handle to the module whose exports are to be enumerated.

pContext
: Program specific context that will be passed to *pfExport*.

pfExport
: Callback function to be called once per symbol exported from module.

Return value

`TRUE` if module exports are enumerated; otherwise `FALSE`.

Error codes

The function sets one of the following error codes, as appropriate. The error code may be retrived after the function has returned by calling `GetLastError`.

ERROR_BAD_EXE_FORMAT
: The MZ header of specified module is invalid.

ERROR_EXE_MARKED_INVALID
: The NT COFF header of the specified module is invalid.

ERROR_INVALID_EXE_SIGNATURE
: The NT COFF header of the specified module has an invalid signature.

Related Samples

Disas, Dumpe, Disas, Einst, Tracelnk, Tracereg.

PF_DETOUR_ENUMERATE_EXPORT_CALLBACK

Pointer to function called once for each export enumerated by DetourEnumerateExports.

Definition

```
BOOL EnumerateExportCallback(
    _In_opt_ PVOID pContext,
    _In_     ULONG nOrdinal,
    _In_opt_ LPCSTR pszName,
    _In_opt_ PVOID pCode
    );
```

Parameters

pContext
: Umodified program specific context pointer passed as pContext argument to DetourEnumerateExports.

nOrdinal
: Ordinal of export function.

pszName
: Name of export function.

pCode
: Pointer to code implementing the function.

Return value

`TRUE` to continue enumeration of export symbols or `FALSE` to abort enumeration.

DetourEnumerateImports

Enumerate imports from a module.

Definition

```
BOOL DetourEnumerateImports(
    _In_opt_ HMODULE hModule,
    _In_opt_ PVOID pContext,
    _In_opt_ PF_DETOUR_IMPORT_FILE_CALLBACK pfImportFile,
    _In_opt_ PF_DETOUR_IMPORT_FUNC_CALLBACK pfImportFunc
    );
```

Parameters

hModule
: The handle to the module whose imports are to be enumerated.

pContext
: Program specific context that will be passed to *pfImportFile* and *pfImportFunc*.

pfImportFile
: Callback function to be called once per file imported by module.

pfImportFunc
: Callback function to be called once per function imported by module.

Return value

`TRUE` if module imports are enumerated; otherwise `FALSE`.

Error codes

The function sets one of the following error codes, as appropriate. The error code may be retrived after the function has returned by calling `GetLastError`.

ERROR_BAD_EXE_FORMAT
: The MZ header of specified module is invalid.

ERROR_EXE_MARKED_INVALID
: The NT COFF header of the specified module is invalid.

ERROR_INVALID_EXE_SIGNATURE
The NT COFF header of the specified module has an invalid signature.

Remarks

If you need a pointer into the Import Address Table ("IAT"), you should use the DetourEnumerateImportsEx.

Related Samples

Tracebld.

PF_DETOUR_IMPORT_FILE_CALLBACK

Pointer to function called once for each file enumerated by DetourEnumerateImports.

Definition

```
BOOL ImportFileCallback(
    _In_opt_ PVOID pContext,
    _In_opt_ HMODULE nOrdinal,
    _In_opt_ LPCSTR pszName
    );
```

Parameters

pContext
: Umodified program specific context pointer passed as pContext argument to DetourEnumerateImports.

hModule
: Module handle within the process of the imported file. NULL to indicate end of enumeration.

pszName
: Name of imported file. NULL to indicate end of enumeration.

Return value

`TRUE` to continue enumeration of import files or `FALSE` to abort enumeration.

PF_DETOUR_IMPORT_FUNC_CALLBACK

Pointer to function called once for each function enumerated by DetourEnumerateImports.

Definition

```
BOOL ImportFuncCallback(
    _In_opt_ PVOID pContext,
    _In_     ULONG nOrdinal,
    _In_opt_ PCSTR pszName,
    _In_opt_ PVOID pvFunc
    );
```

Parameters

pContext
: Umodified program specific context pointer passed as pContext argument to DetourEnumerateImports.

nOrdinal
: Ordinal of imported function. 0 if the import is by name.

pszName
: Name of imported function. NULL if the import is by ordinal.

pvFunc
: Pointer to code implementing the function (or less commonly, data). NULL if the end of the module.

Return value

`TRUE` to continue enumeration of import functions or `FALSE` to abort enumeration.

DetourEnumerateImportsEx

Enumerate imports from a module.

Definition

```
BOOL DetourEnumerateImportsEx(
    _In_opt_ HMODULE hModule,
    _In_opt_ PVOID pContext,
    _In_opt_ PF_DETOUR_IMPORT_FILE_CALLBACK pfImportFile,
    _In_opt_ PF_DETOUR_IMPORT_FUNC_CALLBACK_EX pfImportFunc
    );
```

Parameters

hModule
: The handle to the module whose imports are to be enumerated.

pContext
: Program specific context that will be passed to *pfImportFile* and *pfImportFunc*.

pfImportFile
: Callback function to be called once per file imported by module.

pfImportFunc
: Callback function to be called once per function imported by module.

Return value

`TRUE` if module imports are enumerated; otherwise `FALSE`.

Error codes

The function sets one of the following error codes, as appropriate. The error code may be retrived after the function has returned by calling `GetLastError`.

ERROR_BAD_EXE_FORMAT
: The MZ header of specified module is invalid.

ERROR_EXE_MARKED_INVALID
: The NT COFF header of the specified module is invalid.

ERROR_INVALID_EXE_SIGNATURE
The NT COFF header of the specified module has an invalid signature.

Remarks

DetourEnumerateImports and DetourEnumerateImportsEx are very similar. DetourEnumerateImports's callback receives a pointer to the code (or less commonly, data) pointed to by the Import Address Table ("IAT"). DetourEnumerateImportsEx's callback receives a pointer into the IAT.

Related Samples

Tracebld.

PF_DETOUR_IMPORT_FUNC_CALLBACK_EX

Pointer to function called once for each entry in the IAT enumerated by DetourEnumerateImportsEx. This is similar to PF_DETOUR_IMPORT_FUNC_CALLBACK except here the last parameter points to the entry in the IAT.

Definition

```
BOOL ImportFuncCallbackEx(
    _In_opt_ PVOID pContext,
    _In_     ULONG nOrdinal,
    _In_opt_ PCSTR pszName,
    _In_opt_ PVOID *pvFunc
    );
```

Parameters

pContext
: Umodified program specific context pointer passed as pContext argument to DetourEnumerateImportsEx.

nOrdinal
: Ordinal of imported function. 0 if the import is by name.

pszName
: Name of imported function. NULL if the import is by ordinal.

pvFunc
: Pointer to the address within the Import Address Table ("IAT") for the function (or less commonly, data). NULL if the end of the module.

Return value

`TRUE` to continue enumeration of import functions or `FALSE` to abort enumeration.

DetourEnumerateModules

Enumerate the PE binaries in a process.

Definition

```
HMODULE DetourEnumerateModules(
    _In_opt_ HMODULE hModuleLast
    );
```

Parameters

hModuleLast
: The handle of the last module enumerated. Pass NULL to start enumeration for the beginning of the process.

Return value

Handle to the next module loaded in a process.

Remarks

`DetourEnumerateModules` enumerates all of the PE binaries loaded into a process. Once a module has been enumerated, its entry point can be located with the DetourGetEntryPoint API, its exports can be enumerated with the DetourEnumerateExports API, and its payloads can be found using the DetourFindPayload API.

Related Samples

Disas, Einst, Tracebld, Tracelnk, Tracereg.

DetourFindFunction

Find the address of a target function by name.

Definition

```
PVOID DetourFindFunction(
    _In_ LPCSTR pszModule,
    _In_ LPCSTR pszFunction
    );
```

Parameters

pszModule
The path of the DLL or binary in which the function should be found.
pszFunction
The name of the function to be found.

Return value

If successful, returns the address of the function *pszFunction*; otherwise, returns `NULL`.

Remarks

`DetourFindFunction` tries to retrieve a function pointer for a named function through the dynamic linking export tables of the named module and then, if that fails, through debugging symbols using the DbgHelp APIs if available.

`DetourFindFunction` uses the DbgHelp APIs to access debug symbols. It dynamically links to the DbgHelp APIs by dynamically loading `DBGHELP.DLL`.

If your program can't find debug symbols that you know are available, the most likely cause is that system either can't find `DBGHELP.DLL` or the version of `DBGHELP.DLL` loaded by the system is broken. This can happen on 64-bit machines (X64 and IA64) where the system will often load a 32-bit version of `DBGHELP.DLL` even though you have written 64-bit code. Some 32-bit versions of

Windows also included a non-functional `DBGHELP.DLL` stub.

You can test your version of `DBGHELP.DLL` by using the `SymTest` program from the FindFunc. directory..

Related Samples

Cping, Excep. FindFunc.

DetourFindPayload

Return the address of the specified payload within a module.

Definition

```
_Writable_bytes_(*pcbData)
_Readable_bytes_(*pcbData)
_Success_(return != NULL)
PVOID DetourFindPayload(
    _In_opt_ HMODULE hModule,
    _In_     REFGUID rguid,
    _Out_    DWORD * pcbData
    );
```

Parameters

hModule
: Module holding the payload specified payload.

rguid
: GUID of the specified payload.

pcbData
: Variable to receive the size in bytes of the specified payload.

Return value

Pointer to the specified payload or NULL if the payload doesn't exist.

Error codes

The function sets one of the following error codes if it was unable to search the module for the target payload. The error code may be retrived after the function has returned by calling `GetLastError`.

ERROR_BAD_EXE_FORMAT
: The MZ header of specified module is invalid.

ERROR_EXE_MARKED_INVALID
: The NT COFF header of the specified module is invalid.

ERROR_INVALID_EXE_SIGNATURE
The NT COFF header of the specified module has an invalid signature.

Remarks

`DetourFindPayload` returns the address of the specified payload within a module. Payloads can either be created at compile link time, see the Einst, or can be inserted it an existing binary using the DetourBinarySetPayload API.

For more information on binary editing with Detours and paylods, see Payloads and DLL Import Editing in the Detours Overview.

Related Samples

Einst, Tracebld.

DetourGetContainingModule

Find the PE binary in a process containg a known function.

Definition

```
HMODULE DetourGetContainingModule(
    _In_ PVOID vpAddr
    );
```

Parameters

pvAddr
: Address of a function in the process.

Return value

Handle to the containing module or NULL if the address doesn't reside in a loaded PE binary.

DetourGetEntryPoint

Return the entry point for a module..

Definition

```
PVOID DetourGetEntryPoint(
    _In_opt_ HMODULE hModule
    );
```

Parameters

hModule
The handle of the module to which the entry point is desired.

Return value

Returns the entry point for the module, if found; otherwise, returns `NULL`.

Error codes

The function sets one of the following error codes, as appropriate. The error code may be retrived after the function has returned by calling `GetLastError`.

ERROR_BAD_EXE_FORMAT
The MZ header of specified module is invalid.
ERROR_EXE_MARKED_INVALID
The NT COFF header of the specified module is invalid.
ERROR_INVALID_EXE_SIGNATURE
The NT COFF header of the specified module has an invalid signature.

Remarks

`DetourGetEntryPoint` returns the entry point for a module. For a `.EXE` file, the entry point is the start of the code for the runtime startup runtimes. For a `.DLL file`, the entry point is the start of the code for the `DllMain` function.

The Slept sample shows how to capture program execution after DLL initialization by detouring the entry point of a program.

Related Samples

Disas, Dumpe, Dumpe, Tracebld, Slept.

DetourGetModuleSize

Return the load size of a module.

Definition

```
ULONG DetourGetModuleSize(
    _In_ HMODULE hModule
    );
```

Parameters

hModule
The handle to the module whose load size is desired.

Return value

Returns the size of the module in bytes, if it can be determined; otherwise, returns 0.

Error codes

The function sets one of the following error codes, as appropriate. The error code may be retrived after the function has returned by calling `GetLastError`.

ERROR_BAD_EXE_FORMAT
The MZ header of specified module is invalid.
ERROR_EXE_MARKED_INVALID
The NT COFF header of the specified module is invalid.
ERROR_INVALID_EXE_SIGNATURE
The NT COFF header of the specified module has an invalid signature.

Related Samples

Disas, Tracebld.

DetourIsHelperProcess

Check if the current process is a helper process or a target process.

Definition

```
BOOL DetourIsHelperProcess(VOID);
```

Return value

Returns true if this process is a helper process.

Returns false if this process is a target process.

Remarks

When creating a 32-bit target process from a 64-bit parent process or creating a 64-bit target process from a 32-bit parent process, the DetourCreateProcessWithDllEx API must create a temporary helper process. It loads a copy of the user-supplied DLL into the helper process using the `rundll32.exe` mechanism. The user-supplied DLL should call `DetourIsHelperProcess` within its `DllMain` function to determine if it has been loaded into a helper process or into a target process.

When a user-supplied DLL is loaded into a helper process, it **must not** detour any functions. Instead, it should perform no operations in `DllMain`. The user-supplied DLL must also export the DetourFinishHelperProcess API as its Ordinal 1 export function.

For more information, see Detouring 32-bit and 64-bit Processes.

Related Samples

Simple, Simple, Slept, Traceapi, Tracebld, Tracelnk, Tracemem, Tracereg, Traceser, Tracetcp, Tryman.

DetourFinishHelperProcess

Finishes updating a target process from the helper process.

Definition

```
VOID CALLBACK DetourFinishHelperProcess(
    _In_ HWND,
    _In_ HINSTANCE,
    _In_ LPSTR,
    _In_ INT);
```

Remarks

When creating a 32-bit target process from a 64-bit parent process or creating a 64-bit target process from a 32-bit parent process, the `DetourCreateProcessWithDllEx` API must create a temporary helper process. It loads a copy of the user-supplied DLL into the helper process using the `rundll32.exe` mechanism. `Rundll32.exe` will call DLL's Ordinal 1 export function. The `DetourFinishHelperProcess` API must be set as the DLL's Ordinal 1 export function.

Within its `DllMain` function, a user-supplied DLL can determine if the process is a helper process or a target process by calling the DetourIsHelperProcess API.

For more information, see Detouring 32-bit and 64-bit Processes.

Related Samples

Simple, Simple, Slept, Traceapi, Tracebld, Tracelnk, Tracemem, Tracereg, Traceser, Tracetcp, Tryman.

DetourRestoreAfterWith

Restore the contents in memory import table after a process was started with DetourCreateProcessWithDllEx or DetourCreateProcessWithDlls. .

Definition

```
BOOL DetourRestoreAfterWith(VOID);
```

Return value

Returns true if the necessary payload was found and the restore succeeded.

Error codes

The function sets one of the following error codes if it was unable to find the necessary payload or restore the import table. The error code may be retrived after the function has returned by calling `GetLastError`.

ERROR_MOD_NOT_FOUND
: Could not find the necessary payload.

Remarks

The `DetourCreateProcessWithDllEx` API modifies the in-memory import table of the target PE binary program in the new process it creates. For correct application compatibilty, the changes to the import table should be removed before the application runs. To remove these changes, `DetourCreateProcessWithDllEx` copies relevant reversal data into a payload in the target process using the `DetourCopyPayloadToProcess` API. When called in the target process, `DetourRestoreAfterWith` searches for the necessary payloaded and restores the contents of the import table.

For correct results, `DetourRestoreAfterWith` should be called in the PROCESS_ATTACH portion of the DllMain function of the DLL loaded into the target process.

Related Samples

Simple, Simple, Slept, Traceapi, Tracebld, Tracelnk, Tracemem, Tracereg, Traceser, Tracetcp, Tryman.

DetourSetIgnoreTooSmall

Enable or disable transaction abort on a failure to attach or detach an individual detour function.

Definition

```
BOOL DetourSetIgnoreTooSmall(
    _In_ BOOL fIgnore
    );
```

Return value

Returns true if Detours was previously ignoring failures to detour target functions too smal for detouring.

Parameters

fIgnore
: Specifies whether to ignore functions that are too small to detour. If this parameter is set to **TRUE**, these functions will be ignored if encountered. If this parameter is set to **FALSE**, then encountering a function too small to be detoured will cause DetourTransactionCommit to fail.

Remarks

`DetourSetIgnoreTooSmall` sets the flag to determine if failure to detour a target function that is too small for detouring is sufficient error to cause abort of the current detour transaction.

For more information on using Detours to intercept function calls, see Interception of Binary Functions or Using Detours in the Detours Overview.

Related Samples

Traceapi.

DetourSetRetainRegions

Force Detours to retain trampoline allocation regions even after the trampolines have been released.

Definition

```
BOOL DetourSetRetainRegions(
    _In_ BOOL fRetain
    );
```

Return value

Returns true if Detours was previously retaining unused trampoline regions.

Parameters

fRetain
: Specifies whether trampoline memory allocation regions should be retained (and reused) after all of the trampolines in the region have been released. If this parameter is set to **TRUE**, these regions will be retained. If this parameter is set to **FALSE**, then region will not be retained.

Remarks

Detours allocated trampolines from contiguous regions of 64KB of memory. By default, these regions will be returned to the OS when all of the trampolines in the region have been release (detached). However, in some cases, such as when a program frequently attaches, detaches, and reattaches, it may be desirable to retain the memory regions.

Detours releases prior to version 3.0 always retained trampoline regions. For backward compatibility, some programs may want to force this deprecated behavior by calling `DetourSetRetainRegions(TRUE)`.

DetourSetSystemRegionLowerBound

Set the lower bound of the region of memory that cannot be used for trampolines because it is reserved for system DLLs.

Definition

```
PVOID DetourSetSystemRegionLowerBound(
    _In_ PVOID pSystemRegionLowerBound
    );
```

Return value

Returns the previous lower bound value.

Parameters

pSystemRegionLowerBound
: Specifies the lower bound of the system region into which Detours must avoid placing trampolines.

Remarks

The DetourAttach and DetourAttachEx APIs allocate a trampoline for each detoured function. To avoid fragmenting memory, Detours attempts to create the region from which it allocates trampoline as close as possible to the code being detoured. In some circumstances, trampoline region can collides with memory that is silently set aside by the OS or the application for DLLs that are loaded later. When this happens, the application continues to bahave correctly, but the OS can be forced to relocate one or more DLLs to another location, which increase the virtual memory required for the process.

To avoid these DLL-relocation collisions, Detours is programmed to avoid placing any trampolines in region of memory called the "system region". By default, this region is from 0x70000000 to 0x80000000. Call DetourSetSystemRegionLowerBound and DetourSetSystemRegionUpperBound and to change the region. For example, one might increase the region if the CLR

is loaded into a process after system DLLs are detoured.

For more information on using Detours to intercept function calls, see Interception of Binary Functions or Using Detours in the Detours Overview.

Related Samples

Region.

DetourSetSystemRegionUpperBound

Set the lower bound of the region of memory that cannot be used for trampolines because it is reserved for system DLLs.

Definition

```
PVOID DetourSetSystemRegionUpperBound(
    _In_ PVOID pSystemRegionUpperBound
    );
```

Return value

Returns the previous upper bound value.

Parameters

pSystemRegionUpperBound
: Specifies the upper bound of the system region into which Detours must avoid placing trampolines.

Remarks

The DetourAttach and DetourAttachEx APIs allocate a trampoline for each detoured function. To avoid fragmenting memory, Detours attempts to create the region from which it allocates trampoline as close as possible to the code being detoured. In some circumstances, trampoline region can collides with memory that is silently set aside by the OS or the application for DLLs that are loaded later. When this happens, the application continues to bahave correctly, but the OS can be forced to relocate one or more DLLs to another location, which increase the virtual memory required for the process.

To avoid these DLL-relocation collisions, Detours is programmed to avoid placing any trampolines in region of memory called the "system region". By default, this region is from 0x70000000 to 0x80000000. Call DetourSetSystemRegionLowerBound and DetourSetSystemRegionUpperBound and to change the region. For example, one might increase the region if the CLR

is loaded into a process after system DLLs are detoured.

For more information on using Detours to intercept function calls, see Interception of Binary Functions or Using Detours in the Detours Overview.

Related Samples

Region.

DetourTransactionAbort

Abort the current transaction.

Definition

```
LONG DetourTransactionAbort(VOID);
```

Return value

Returns NO_ERROR if the pending transaction was completely aborted; otherwise, returns an error code.

Error codes

ERROR_INVALID_OPERATION
No pending transaction exists.

Remarks

`DetourTransactionAbort` aborts the current transaction created with DetourTransactionBegin. Aborting a transaction reverse the effects of any calls to the DetourAttach, DetourAttachEx, DetourDetach, or DetourUpdateThread APIs made within the transaction.

For more information on using Detours to intercept function calls, see Interception of Binary Functions or Using Detours in the Detours Overview.

DetourTransactionBegin

Begin a new transaction for attaching or detaching detours.

Definition

```
LONG DetourTransactionBegin(VOID);
```

Return value

Returns NO_ERROR if successful; otherwise returns ERROR_INVALID_OPERATION.

Error codes

ERROR_INVALID_OPERATION
A pending transaction alrady exists.

Remarks

`DetourTransactionBegin` begins a new transaction for attaching or detaching detours.

After beginning a transaction, a program calls the DetourAttach or DetourAttachEx API to attach a detour to a target function, calls the DetourDetach API to detach a detour from a target function, or calls the DetourUpdateThread API to include include a thread in the transaction update.

The attach, detach, and thread operations do not take effect until the program commits the transaction using the DetourTransactionCommit or DetourTransactionCommitEx API. Alternatively, the program can abort the transaction using the DetourTransactionAbort API.

For more information on using Detours to intercept function calls, see Interception of Binary Functions or Using Detours in the Detours Overview.

Related Samples

Commem, Cping, Dtest, Excep, FindFunc, Member, Simple, Slept, Traceapi, Tracebld, Tracelnk, Tracemem, Tracereg, Traceser, Tracetcp, Tryman.

DetourTransactionCommit

Commit the current transaction.

Definition

```
LONG DetourTransactionCommit(VOID);
```

Return value

Returns NO_ERROR if successful; otherwise, returns an error code.

Error codes

ERROR_INVALID_DATA
Target function was changed by third party between steps of the transaction.

ERROR_INVALID_OPERATION
No pending transaction exists.

Other
Error code returned by API within DetourAttach, DetourAttachEx, or DetourDetach that caused transaction to fail.

Remarks

`DetourTransactionCommit` commits the current transaction created with DetourTransactionBegin. Commiting a transaction make all updates specified in any calls to the DetourAttach, DetourAttachEx, DetourDetach, or DetourUpdateThread APIs within the transaction.

For more information on using Detours to intercept function calls, see Interception of Binary Functions or Using Detours in the Detours Overview.

Related Samples

Commem, Cping, Dtest, Excep, FindFunc, Member, Simple, Slept, Traceapi, Tracebld, Tracelnk, Tracemem, Tracereg, Traceser, Tracetcp, Tryman.

DetourTransactionCommitEx

Commit the current transaction.

Definition

```
LONG DetourTransactionCommitEx(
    _Out_opt_ PVOID ** pppFailedPointer
    );
```

Parameters

pppFailedPointer
: Variable to receive the target pointer passed to the DetourAttach, DetourAttachEx, or DetourDetach call that caused the latest transaction to fail.

Return value

Returns NO_ERROR if successful; otherwise, returns an error code.

Error codes

ERROR_INVALID_DATA
: Target function was changed by third party before the transaction could complete.

ERROR_INVALID_OPERATION
: No pending transaction exists.

Other Codes
: Error code returned by API within DetourAttach, DetourAttachEx, or DetourDetach that caused transaction to fail.

Remarks

`DetourTransactionCommitEx` commits the current transaction created with DetourTransactionBegin. Commiting a transaction make all updates specified in any calls to the DetourAttach, DetourAttachEx, DetourDetach,

or DetourUpdateThread APIs within the transaction.

For more information on using Detours to intercept function calls, see Interception of Binary Functions or Using Detours in the Detours Overview.

Related Samples

Traceapi,

DetourUpdateThread

Enlist a thread for update in the current transaction.

Definition

```
LONG DetourUpdateThread(
    _In_ HANDLE hThread
    );
```

Parameters

hThread
The handle of the thread to be updated with the pending transaction.

Return value

Returns NO_ERROR if successful; otherwise, returns an error code.

Error codes

ERROR_NOT_ENOUGH_MEMORY
Not enough memory to record identity of thread.

Remarks

`DetourUpdateThread` enlists the specified thread for update when the current transaction, opened by the DetourTransactionBegin API, commits.

When a detour transaction commmits, Detours insures that all threads enlisted in the transcation via the `DetourUpdateThread` API are updated if their instruction pointer lies within the rewritten code in either the target function or the trampoline function.

Threads not enlisted in the transaction are not updated when the transaction commits. As a result, they may attempt to execute an illegal combination of old and new code.

For more information on using Detours to intercept function calls, see Interception of Binary Functions or Using Detours in the Detours Overview.

Related Samples

Commem, Cping, Dtest, Excep, FindFunc, Member, Simple, Slept, Traceapi, Tracebld, Tracelnk, Tracemem, Tracereg, Traceser, Tracetcp, Tryman.

Building The Samples

To build the sample applications, type `nmake` in the samples directory. Note that you must build the setdll and syslog samples in order to use many of the other sample programs.

Each of the sample directories has a test, which can be invoked by typing `nmake test`, to demonstrate the usage of the sample. With very few exceptions, all of the .exe programs also accept a `/?` command to display a usage message.

The trace samples log their output through the syelogd.exe daemon and hook `CreateProcessW` to load themselves into any child processes. For example, typing `withdll -d:traceapi.dll cmd.exe` will create a command shell under which all processes log their API calls through traceapi.dll.

Detours includes the following samples:

- Commem
- Cping
- Disas
- Dtest
- Dumpe
- Dumpi
- Einst
- Excep
- FindFunc
- Impmunge
- Member
- Region
- Setdll
- Simple
- Slept
- Syelog
- Traceapi
- Tracebld
- Tracelnk
- Tracemem
- Tracereg

- Traceser
- Tracetcp
- Tryman
- Withdll

Commem

Demostrates how to detour a member function of a COM interface.

Uses

DetourAttach, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

Cping

Detours multiple functions in the DCOM/RPC stack to measure the overhead of sending DCOM messages.

Uses

DetourAttach, DetourFindFunction, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

Disas

Tests the Detours disassembler tables.

Uses

DetourEnumerateExports, DetourEnumerateModules, DetourGetEntryPoint, DetourGetModuleSize.

Dtest

Detours the Win32 `Sleep` function and a private function. The private function is first detoured, then detoured recursively 3 times using the DetourAttach API.

Uses

DetourAttach, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

Dumpe

Dumps the list of all functions exported from a binary.

Uses

DetourEnumerateExports, DetourGetEntryPoint.

Dumpi

Dumps the list of all functions imported by a binary.

Uses

DetourBinaryClose, DetourBinaryEditImports, DetourBinaryOpen.

Einst

Find payloads compiled into binary files.

Uses

DetourEnumerateModules, DetourFindPayload.

Excep

Uses a first-chance exception filter to toggle VM permissions on a page; enabling writes after catching the first write to a page.

Uses

DetourAttach, DetourFindFunction, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

FindFunc

Demostrates how to detour a function using DetourFindFunction to find the function using debug symbols.

Uses

DetourAttach, DetourDetach, DetourFindFunction, DetourFinishHelperProcess, DetourIsHelperProcess, DetourRestoreAfterWith, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

Impmunge

Modifies all of the entries in a binary's imports table.

Uses

DetourBinaryClose, DetourBinaryEditImports, DetourBinaryOpen, DetourBinaryWrite.

Member

Demostrates how to detour a class member function.

Uses

DetourAttach, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

Region

Demostrates how to change the region of memory off limits for trampolines.

Uses

DetourAttach, DetourTransactionBegin, DetourTransactionCommit, DetourSetSystemRegionLowerBound, DetourSetSystemRegionUpperBound, DetourUpdateThread.

Setdll

Add a DLL to the import table of any binary (a `.DLL` or `.EXE` for example). Use `setdll.exe` to attach one of the sample DLLs to an application .EXE file.

Note: The target binary will fail to load if the target DLL does not contain a exported function with ordinal #1. For more information, see the DetourBinaryEditImports API.

Related Samples

Withdll.

Uses

DetourBinaryClose, DetourBinaryEditImports, DetourBinaryOpen, DetourBinaryResetImports, DetourBinaryWrite.

Simple

Simplest example of a Detours-based DLL which modifies and adds functionality to a Windows API. Modifies the `Sleep` API to record the number of ticks spent sleeping.

The Simple example is described in more detail in Using Detours.

Uses

DetourAttach, DetourDetach, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

Slept

More ellaborate version of the simple sample. Demonstrates detouring both static and dynamic functions. Also demonstrates how to capture program execution after DLL initialization by detouring the programs entry point.

Uses

DetourAttach, DetourCodeFromPointer, DetourDetach, DetourGetEntryPoint, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

Syelog

System event logging library and service. All of the tracing samples connect to `syelogd.exe` through a named pipe. Syelogd outputs tracing information to the console. You must run `syelogd.exe` in a seperate window in order to see the output from any of the following tracing DLLs: Traceapi, Tracelnk, Tracemem, Tracereg, Traceser, or Tracetcp.

Uses

Uses none of the Detours APIs.

Traceapi

Win32 API tracing sample. Detours and prints tracing statements for 1401 Win32 API functions. Output from the trace is logged to the syelogd.exe deamon.

Uses

DetourAttach, DetourCreateProcessWithDllEx, DetourDetach, DetourFinishHelperProcess, DetourIsHelperProcess, DetourRestoreAfterWith, DetourSetIgnoreTooSmall, DetourTransactionBegin, DetourTransactionCommit, DetourTransactionCommitEx, DetourUpdateThread.

Tracebld

Traces the file access patterns of a process and all of its children. Unlike the other tracing samples, Tracebld is entirely self-contained. It includes a parent process that initiates a child process with instrumentation and aggregates the results for the child and its children. Output from the children is delivered to the parent process via a named pipe created by the parent.

Uses

DetourAttach, DetourAttachEx, DetourCodeFromPointer, DetourCopyPayloadToProcess, DetourCreateProcessWithDllEx, DetourDetach, DetourEnumerateImports, DetourEnumerateModules, DetourFindPayload, DetourGetEntryPoint, DetourGetModuleSize, DetourFinishHelperProcess, DetourIsHelperProcess, DetourRestoreAfterWith, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

TraceInk

Traces all calls to the Windows dynamic linking APIs. Output from the trace is logged to the syelogd.exe deamon.

Uses

DetourAttach, DetourDetach, DetourEnumerateModules, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

Tracemem

Traces all calls to the Windows `HeapAlloc` API. Output from the trace is logged to the syelogd.exe deamon.

Uses

DetourAttach, DetourCreateProcessWithDllEx, DetourDetach, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

Tracereg

Traces activity through the registry APIs. Output from the trace is logged to the syelogd.exe deamon.

Uses

DetourAttach, DetourCreateProcessWithDllEx, DetourDetach, DetourEnumerateModules, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

Traceser

Traces activity through the serial ports (com1 or com2). Output from the trace is logged to the syelogd.exe deamon.

Uses

DetourAttach, DetourCreateProcessWithDllEx, DetourDetach, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

Tracetcp

Traces activity through WinSock TCP APIs. Output from the trace is logged to the syelogd.exe deamon.

Uses

DetourAttach, DetourDetach, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

Traceapi

Demonstration of using helper processes to hook both 32-bit and 64-bit target processes.

Uses

DetourAttach, DetourCreateProcessWithDllEx, DetourDetach, DetourFinishHelperProcess, DetourIsHelperProcess, DetourRestoreAfterWith, DetourTransactionBegin, DetourTransactionCommit, DetourUpdateThread.

Withdll

Demonstrates how to use the DetourCreateProcessWithDlls API to load a detour DLL into a new process without modifying the target application. Calls `CreateProcess` and loads a named DLL into the target process.

Note: The new process will fail to start if the target DLL does not contain a exported function with ordinal #1. For more information, see the DetourCreateProcessWithDlls API.

Related Samples

Setdll.

Uses

DetourCreateProcessWithDlls .

Detours Frequently Asked Questions (FAQ)

This page contains a list of questions frequently asked about Detours. The questions are grouped by general topic and area of interest.

Compatibility

Is Detours compatible with Windows 8?

Yes. Detours is fully compatible with Windows 8 desktop and server applications. While Detours can be used in the development and testing of Window Store apps, new Windows Store apps for Windows 8 can not ship with Detours.

Why can't my Windows Store app for Windows 8 include Detours?

Windows Store apps may use only a subset of the Win32 API. Detours requires several Win32 APIs that are forbidden in for Windows App Certification. Forbidden APIs used by Detours include VirtualAlloc, VirtualProtect, and FlushInstructionCache.

Is Detours compatible with Windows 95, Windows 98, or Windows ME?

No. Detours is compatible only with the Windows NT family of operating systems: Windows NT, Windows XP, and Windows Server 2003, etc. Detours does not work on the Windows 9x family of operating systems because they have a primitive virtual memory system.

Compiling with Detours Code

How do I do thing X with Detours?

Look in the Detours Samples. The Detours Samples are quite extensive. It is likely that anything you want to accomplish with Detours is covered in one of the included samples.

Where can I find `detours.lib` and `detours.h`?

You need to build a version of `detours.lib` for your C/C++ compiler in the `detours/src` directory by typing `nmake` either in the `detours` directory or in the `detours/src` directory.

Running with Detours

Why don't I see any calls to my detour of `malloc`?

Probably because the target program is not using the `malloc` function you detoured.

Standard library functions like `malloc` can be linked with a program either statically, from one of the `libc*.lib` libraries, or dynamically, from one of the `msvcrt*.dll` libraries. When statically linked, a program receives its own private version of the standard library functions. When dynamically linked, a program shares version of the standard library functions in a DLL. If you detour your private version of the function, or if the target program uses its own private version of the function, your detour won't be called by the target program.

Why is Detours packaged as a static library (`detours.lib`) and not as a dynamic link library (say `detours.dll`)?

Packaging Detours as a statical library minimizes the risk that you will accidentally detour a function required by the Detours package itself and reduces versioning problems. Note that Detours adds only about 16KB when statically linked with your code.

Do I still need to use `detoured.dll`?

No, the `detoured.dll` marker file was removed in Detours 3.0. Before Detours 3.0, this file was used as marker to guide Microsoft technical support personnel and tools, like Windows Error Reporting, by helping them quickly determine that a process has been altered by the Detours package. Advances in Windows OCA in Windows 7 removed the need for this marker as Windows 7 maintains a list of DLL that have been unloaded from a process. Microsoft can not guarantee nor support in any way, the modification of Microsoft binaries by third parties. Nor can Microsoft support, in any way, an application that contains Microsoft binaries modified by third parties. This includes in-memory modification using the Detours package.

How can I debug the startup of my detour DLL?

The Windbg can single step or break on exceptions in process startup. Windbg is available in the "Debugging Tools for Windows" download from on www.microsoft.com. For example, you can use the command line:

```
windbg -o withdll.exe -d:mydll.dll myexe.exe
```

Why does my code act differently under a debugger?

Debuggers insert breakpoints by replacing function code with break instructions. For example, on the X86 and X64 processors, the debugger will write a 0xCC (int 3) for a breakpoint. If the breakpoint is written before a detour is applied, the Detour library will see the 0xCC instead of the real instructions.

The best way to work around this issue is to ensure that no debugger breakpoints are set on target functions.

Licensing

Can Detours be used in commercial applications?

Yes, with a Detours Professional license. You can purchase Detours Professional from the Microsoft Store.

Bug Reports

How do I report a bug?

Please send detailed bug reports to detours@microsoft.com. Bug reports may be used to fix bugs in future versions of the Detours package. Please include the text "DETOURS BUG REPORT" in the subject line. Within the body of your message, please include the first line from the README.TXT file which contains the full description of the version Detours you are using including the Build number.

Before submitted a bug report, please make every effort to insure that the problem is not an error in your own code or your usage of Detours. The most common sources of user error are covered in this FAQ.

The detours@microsoft.com email address is for bug reports only, it is not a product support line.

DetourGetSizeOfPayloads

Return the size of all payloads within a module.

Definition

```
DWORD DetourGetSizeOfPayloads(
    _In_ HMODULE hModule
    );
```

Parameters

hModule
: The module for which the size of the payloads is to be returned.

Return value

If successful, returns the size in bytes of the payloads within a module; otherwise, returns zero.

Error codes

The function sets one of the following error codes, as appropriate. The error code may be retrived after the function has returned by calling `GetLastError`.

ERROR_INVALID_HANDLE
: The module handle was invalid.

Related Samples

Einst.