



Who the F called me?

A method of obtaining caller module and function names
from a CRT-less & Win32-less injected DLL



Content coverage

What will be covered:

1. Stripping CRT from your PE & Using NTDLL as your sole dependency.
2. Loader translation and API call hierarchy.
3. Caller module name acquisition without symbols.
4. Exception handler record information gathering.
5. Caller function name acquisition without symbols.
6. Use of hooking as a method for anti-(anti-debug) / human friendly inspection.

What, Where, Why?

Contents

Glossary	5
1 Introduction	10
1.1 Background	10
1.2 Aim & Objectives of the Study	12
1.3 Legal, ethical, societal, GDPR, and security considerations	12
2 Literature Review	13
2.1 Windows Internals Part 1	13
2.2 Detours - Microsoft	13
2.3 DetoursNT - wbenny	15
2.4 Lets Create An EDR... And Bypass It! - ethical chaos	16
2.5 RTO-MDI - Red Team Operator, Malware Development , sektor7	17
2.6 Focus and intention	18
3 Project Management	19
3.1 Project timeline & Planning	19
3.2 Environment	19
3.3 Codebase management	19
3.4 Requirements	21
4 Methodology	22
4.1 Windows Defender on-disk detection & detection bypass	23
4.2 Tracing API calls from Win32 to the kernel, One Page at a Time	30
4.3 Hooking MessageBoxW with Microsoft Detours	37
5 Implementation - EDRAgent, The DLL	43
5.1 Intended Audience	43
5.2 _ReturnAddress() compiler intrinsic	43
5.3 Hooking the Win32 API with DetoursNT - wbenny	44
5.4 Finding the caller module name	48
5.5 Finding the calling function name	51
5.6 Putting the pieces together	55
5.7 Bypassing detection	61
6 Critical Review	65
6.1 Project Management Review	65
6.2 Success Evaluation	65
6.3 Revisions	65
6.4 Future Developments/Research	68
7 References	70

4 Methodology	22
4.1 Windows Defender on-disk detection & detection bypass	23
4.2 Tracing API calls from Win32 to the kernel, One Page at a Time	30
4.3 Hooking MessageBoxW with Microsoft Detours	37
5 Implementation - EDRAgent, The DLL	43
5.1 Intended Audience	43
5.2 _ReturnAddress() compiler intrinsic	43
5.3 Hooking the Win32 API with DetoursNT - wbenny	44
5.4 Finding the caller module name	48
5.5 Finding the calling function name	51
5.6 Putting the pieces together	55
5.7 Bypassing detection	61



Wanted to write something cool

Hobby EDR - Kernel-Mode driver to inject dlls into processes somehow
Detect & Block Known Bad such as allocation / execution
Bypass my own efforts and refactor
Grow skillset / get good



The issue

I had the main “bad” Win32/NTAPI functions usermode hooked such as

- Memory allocation
- Process creation
- Memory protection manipulation

But was unsure at how to convert hooked API call into human friendly / understandable output.



1. CRT(UCRT) & win32

CRT?

<https://docs.microsoft.com/en-us/cpp/c-runtime-library/crt-library-features?view=msvc-XXX>

Universal C-runtime (UCRT aka CRT) implementation specific code:

- EH / Debugging
- runtime checks
- STL

Win32?

<https://docs.microsoft.com/en-us/windows/win32/desktop-programming>

C/C++ OS / higher level hardware access APIs:

- MM
- COM / GUI / User input etc
- Provides abstraction from OS specific implementation

TL;DR CRT allows you to screw up gracefully, and win32 allows you to stay portable due to the abstraction & encapsulation it provides



Cleaning dependencies





PE Example 1

```
#include <stdio.h>

void main(void)
{
    printf("PE Example 2\n");
}
```



```
C:\TOOLING\BSIDES_CFP\DetoursNT\Bin\x64\Debug>dumpbin /imports:ucrtbased.dll print_demo_no_clean.exe
Microsoft (R) COFF/PE Dumper Version 14.29.30145.0
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
Dump of file print_demo_no_clean.exe
```

```
File Type: EXECUTABLE IMAGE
```

```
Section contains the following imports:
```

```
ucrtbased.dll
1400201F0 Import Address Table
1400205E0 Import Name Table
0 time date stamp
0 Index of first forwarder reference

        68 __stdio_common_vsprintf_s
2B5 _register_onexit_function
E5 _execute_onexit_table
C2 _crt_atexit
C1 _crt_at_quick_exit
54B terminate
39C _wmakepath_s
3B8 _wsplitpath_s
564 wcsncpy_s
2C2 _seh_filter_dll
528 strcat_s
52C strcpy_s
4D _p_commode
2CE _set_new_mode
B5 _configthreadlocale
2B6 _register_thread_local_exe_atexit_callback
9F _c_exit
A4 _cexit
4A _p_argv
49 _p_argc
2CB _set_fmode
EA _exit
450 exit
175 _initterm_e
174 _initterm
13D _get_initial_narrow_environment
171 _initialize_narrow_environment
B6 _configure_narrow_argv
5B _setusermatherr
2C6 _set_app_type
2C3 _seh_filter_exe
5 _CrtDbgReportW
4 _CrtDbgReport
5C __stdio_common_vfprintf
35 __acrt_iob_func
172 _initialize_onexit_table
```

```
C:\TOOLING\BSIDES_CFP\DetoursNT\Bin\x64\Debug>dumpbin /imports:vcruntime140d.dll print_demo_no_clean.exe
Microsoft (R) COFF/PE Dumper Version 14.29.30145.0
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
Dump of file print_demo_no_clean.exe
```

```
File Type: EXECUTABLE IMAGE
```

```
Section contains the following imports:
```

```
VCRUNTIME140D.dll
140020150 Import Address Table
140020540 Import Name Table
0 time date stamp
0 Index of first forwarder reference

2E __vcrtd_GetModuleFileNameW
1C __current_exception_context
1B __current_exception
25 __std_type_info_destroy_list
9 __C_specific_handler_noexcept
2F __vcrtd_GetModuleHandleW
31 __vcrtd_LoadLibraryExW
8 __C_specific_handler
```



The code execution cannot proceed because VCRUNTIME140.dll was not found. Reinstalling the program may fix this problem.

OK



The code execution cannot proceed because MSVCP140.dll was not found. Reinstalling the program may fix this problem.

OK



PE Example 2

```
#include "phnt/phnt_windows.h"
#include "phnt/phnt.h"
#pragma comment(lib, "ntdllp.lib")

void CustomEntryPoint(void)
{
    DbgPrint("PE Example 2\n");
}
```

This can be cleaned up

print_demo_cleaned Property Pages

Configuration: All Configurations Platform: All Platforms Configuration Manager...

Configuration Properties

- C/C++
- Basic Runtime Checks
- Default

Look for options or switches:

Basic Runtime Checks	Default
----------------------	---------

print_demo_cleaned Property Pages

Configuration: All Configurations Platform: All Platforms Configuration Manager...

Configuration Properties

- C/C++
- SDL checks
- Security Check
- No (/sdl-)
- Disable Security Check (/GS-)

Look for options or switches:

SDL checks	No (/sdl-)
Security Check	Disable Security Check (/GS-)

print_demo_cleaned Property Pages

Configuration: All Configurations Platform: All Platforms Configuration Manager...

Configuration Properties

- Linker
- Additional Dependencies
- ntdll.lib;%(AdditionalDependencies)

Look for options or switches:

Additional Dependencies	ntdll.lib;%(AdditionalDependencies)
-------------------------	-------------------------------------

All Options
Command Line
Manifest Tool

print_demo_cleaned Property Pages

Configuration: All Configurations Platform: All Platforms Configuration Manager...

Configuration Properties

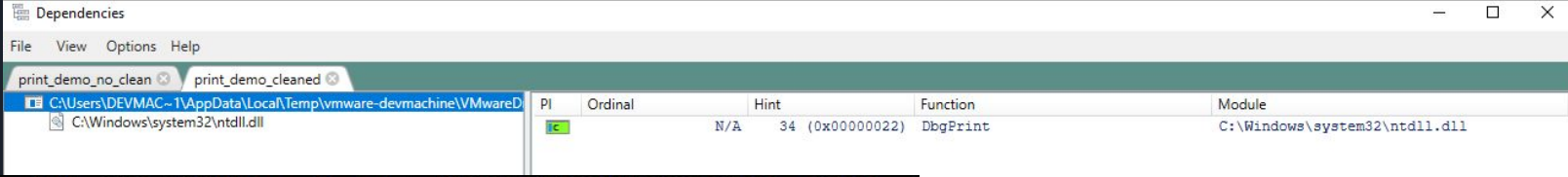
- Linker
- Show Progress
- Display all progress messages (/VERBOSE)

Look for options or switches:

Show Progress	Display all progress messages (/VERBOSE)
---------------	--

All Options

PE Example 2



```
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community>dumpbin /IMPORTS C:\TOOLING\BSIDES_CFP\DetoursNT\DetoursNT\x64\Debug\print_demo_cleaned.exe
Microsoft (R) COFF/PE Dumper Version 14.29.30145.0
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
Dump of file C:\TOOLING\BSIDES_CFP\DetoursNT\DetoursNT\x64\Debug\print_demo_cleaned.exe
```

```
File Type: EXECUTABLE IMAGE
```

```
Section contains the following imports:
```

```
ntdll.dll
140018000 Import Address Table
140018088 Import Name Table
0 time date stamp
0 Index of first forwarder reference

22 DbgPrint
```

```
Summary
```

```
1000 .data
1000 .idata
1000 .msvcjmc
2000 .pdata
2000 .rdata
1000 .reloc
1000 .rsrc
2000 .text
10000 .textbss
```

Function	VirtualAddress			
Virtual Size	Entry point	Subsystem	Subsystem Ver.	Checksum
0x001f5000	0x00000000	0x00000003	10.0	0x001f842b (co

From an analysts perspective

The image displays two side-by-side windows of IDA Pro, comparing the original and cleaned versions of a binary. The left window is titled 'IDA - print_demo_no_clean.exe' and the right window is titled 'IDA - print_demo_cleaned.exe'. Both windows show the IDA View-A, Pseudocode-A, and Hex View-1 panes.

Left Window (Original):

- Function List:** A list of functions is shown on the left. A red circle with the number '1' highlights the function `j_set_fmode`.
- IDA View-A:** The assembly code for `j_set_fmode` is displayed. It includes instructions like `lea rcx, [rsp+458h+var_228]`, `call sub_140014C50`, `test eax, eax`, and `jz short loc_14001480D`.
- Pseudocode-A:** The pseudocode for `j_set_fmode` is shown, including `FileNamed(ModuleHandleW, v2, 260164) == 0` and `j_vcruntimeLoadLibraryExW(L"VCRUNTIME140D.dll");`.
- Hex View-1:** The hex dump for `loc_14001480D` is shown, including `xor edx, edx`, `lea rcx, aMSPDB140 ; "MSPDB140"`, `mov r8d, 0A00h`, `call j_vcruntimeLoadLibraryExW`, `test rax, rax`, and `jnz short loc_14001487D`.

Right Window (Cleaned):

- Function List:** A list of functions is shown on the left. A red circle with the number '2' highlights the function `start_0`.
- IDA View-A:** The assembly code for `start_0` is displayed, including `xor edx, edx`, `lea rcx, [rsp+458h+var_438]`, `mov r8d, 900h`, `call j_vcruntimeLoadLibraryExW`, `test rax, rax`, and `jnz short loc_14001487D`.
- Pseudocode-A:** The pseudocode for `start_0` is shown, including `FileNamed(0i64, v3, 260164) == 0` and `cs:DbgPrint("DBGPRINT: Demo Print statement");`.
- Hex View-1:** The hex dump for `start_0` is shown, including `xor edx, edx`, `lea rcx, [rsp+458h+var_438]`, `mov r8d, 900h`, `call j_vcruntimeLoadLibraryExW`, `test rax, rax`, and `jnz short loc_14001487D`.



Loader translation / contracts

Devil is in the implementation

```
#include <Windows.h>
#include <stdio.h>

int main(void) {
    void * mem = VirtualAlloc(0,
        4096,
        MEM_COMMIT | MEM_RESERVE,
        PAGE_EXECUTE_READWRITE
    );
    printf("Allocated Memory At: %p", mem);
    getchar();
    return 0;
}
```

Dependencies (x64)

File View Options Help

SampleHookDLL x kernel32

```
C:\Windows\System32\kernel32.dll
api-ms-win-core-rtlsupport-l1-1-0.dll -> C:\Windows\system32\ntdll.dll
C:\Windows\system32\ntdll.dll
C:\Windows\system32\kernelbase.dll
api-ms-win-core-processthreads-l1-1-0.dll -> C:\Windows\system32\kernel32.dll
api-ms-win-core-processthreads-l1-1-3.dll -> C:\Windows\system32\kernel32.dll
api-ms-win-core-processthreads-l1-1-2.dll -> C:\Windows\system32\kernel32.dll
api-ms-win-core-processthreads-l1-1-1.dll -> C:\Windows\system32\kernel32.dll
api-ms-win-core-registry-l1-1-0.dll -> C:\Windows\system32\kernel32.dll module loaded correctly
api-ms-win-core-heap-l1-1-0.dll -> C:\Windows\system32\kernelbase.dll
api-ms-win-core-heap-l2-1-0.dll -> C:\Windows\system32\kernelbase.dll
api-ms-win-core-memory-l1-1-1.dll -> C:\Windows\system32\kernelbase.dll
api-ms-win-core-memory-l1-1-0.dll -> C:\Windows\system32\kernelbase.dll
api-ms-win-core-memory-l1-1-2.dll -> C:\Windows\system32\kernelbase.dll
api-ms-win-core-handle-l1-1-0.dll -> C:\Windows\system32\kernelbase.dll
api-ms-win-core-synch-l1-1-0.dll -> C:\Windows\system32\kernelbase.dll
```

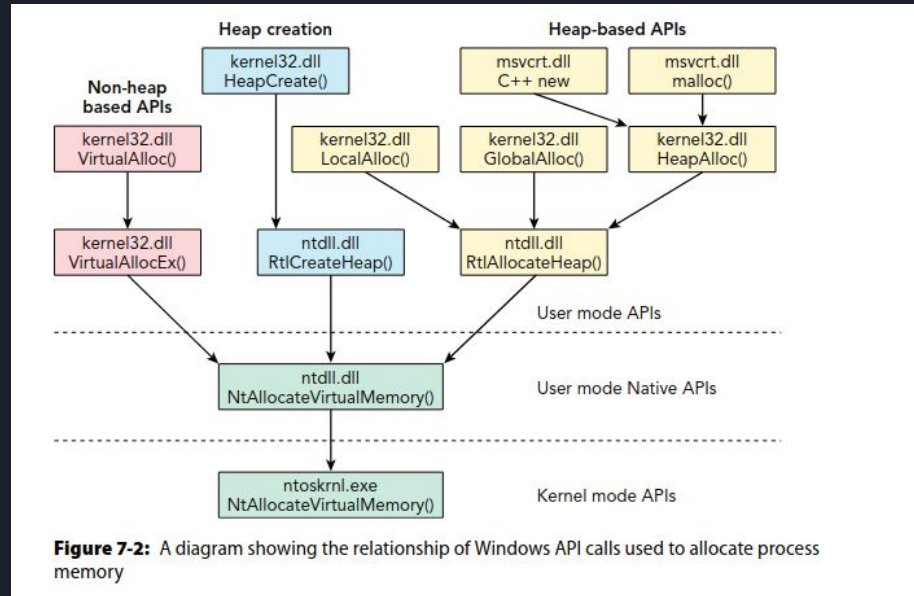


Figure 7-2: A diagram showing the relationship of Windows API calls used to allocate process memory

The Art of Memory Forensics (J. and W. A. 2014) simplifying the apisetmap translation

```
D:\Collections\University\FYP\DetoursNT_DEMO\x64\Debug>apisetparse.exe | findstr /i api-ms-win-core-memory
api-ms-win-core-memory-l1-1-7.dll, kernelbase.dll
```



Api contracts

<https://docs.microsoft.com/en-gb/windows/win32/apiindex/windows-apisets>

<https://lucasg.github.io/2017/10/15/Api-set-resolution/>

<https://docs.microsoft.com/en-gb/windows/win32/apiindex/detect-api-set-availability>

<https://github.com/zodiacon/WindowsInternals/blob/master/APISetMap/APISetMap.cpp>

ApiSetMap from the PEB holds these contracts as a hash table which can be parsed.

Early load

Now the PE depends on solely NTDLL.DLL it is a good base to work off of if we wanted to use a driver to early load KAPC inject / load the PE.
Gives us the freedom but not the portability.

wbenny / injdrv Public

Notifications Fork 220 Star 769

Code Issues 12 Pull requests 1 Actions Projects Wiki Security Insights

master

Go to file Code

About

proof-of-concept Windows Driver for injecting DLL into user-mode processes using APC

injection hooking windows-driver

Readme MIT license 769 stars 39 watching 220 forks

Releases No releases published

wbenny	retarget solution to latest MSVC	on 1 Jun 2021	22
img	rewrite everything	4 years ago	
include	retarget solution to latest MSVC	13 months ago	
src	retarget solution to latest MSVC	13 months ago	
.editorconfig	rewrite everything	4 years ago	
.gitignore	rewrite everything	4 years ago	
.gitmodules	add NT-headers as a submodule	4 years ago	
LICENSE.txt	rewrite everything	4 years ago	
README.md	update README.md	3 years ago	
inj.sln	split injection code into separate project (injlib)	4 years ago	

dennisbabkin / InjectAll Public

Notifications Fork 12 Star 35

Code Issues 2 Pull requests 1 Actions Projects Wiki Security Insights

main

Go to file Code

About

Tutorial that demonstrates how to code a Windows driver to inject a custom DLL into all running processes. I coded it from start to finish using C++ and x86/x64 Assembly language in Microsoft Visual Studio. The solution includes a kernel driver project, a DLL project and a C++ test console project.

dennisbabkin	Fixed missing closing angle bracket	on 2 Aug 2021	24
InjectAll	Fixed missing closing angle bracket	11 months ago	
README.md	Update README.md	13 months ago	

README.md



TL;DR

InjectAll project - dennisbabkin

- Register a driver-supplied callback so that whenever a PE (DLL/EXE) is loaded into memory we can inject a DLL
- Check if the PE is `IsMappedByLdrLoadDll`
- Steal security descriptor of a `KnownDll` e.g. `kernel32.dll`
- Create permanent section (`OBJ_PERMANENT`)
- Allocate `NonPagedPool`
- Create section in the `KnownDll`'s kernel object dir with our DLL to be injected + stolen SD
- allocate our KAPC from `NonPagedPool`

- DLL is a simple assembly DLL so that its position independent and doesn't need reloc



Kernel-mode Asynchronous Procedure Call

specify a callback routine to execute asynchronously...callback to a particular thread

Allows us to have a driver running which injects a DLL into every process running just before kernel32 loads into the process.

<https://docs.microsoft.com/en-us/windows/win32/sync/asynchronous-procedure-calls>

https://dennisbabkin.com/inside_nt_apc/ # great but heavy read

<https://dennisbabkin.com/blog/?t=windows-apc-deep-dive-into-user-mode-asynchronous-procedure-calls>

<https://dennisbabkin.com/blog/?t=depths-of-windows-apc-aspects-of-asynchronous-procedure-call-internals-from-kernel-mode>



Hooking

Microsoft Detours

microsoft / Detours Public

Notif

Code Issues 32 Pull requests 10 Discussions Actions Projects 1 Wiki Security Insights

master 2 branches 1 tag

Go to file Code

About

Detours is a software package for monitoring and instrumenting API calls on Windows. It is distributed in source code form.

- fishjam and fishjam Fix: Handle ERROR_PIPE_CONNECTED error from ConnectNamedPipe... a1dd93f 3 days ago 113 commits
- .github Maintenance: Migrate FabricBot Tasks to Config-as-Code 3 days ago
- examples Fix: Handle ERROR_PIPE_CONNECTED error from ConnectNamedPipe in... 3 days ago

```
p4y10_q36yak@MacBookAir MINGW64 /d/Collections/University/FYP/Source_Files/Detours-4.0.1
$ grep -r -i -e VirtualAlloc -e VirtualProtect 2>/dev/null | grep -i src
src/creatwth.cpp:     PBYTE pbAlloc = (PBYTE)VirtualAllocEx(hProcess, pbAddress, cbAlloc,
src/creatwth.cpp:         DETOUR_TRACE(("VirtualAllocEx(%p) failed: %d\n", pbAddress, GetLastError()));
src/creatwth.cpp:     if (!DetourVirtualProtectSameExecuteEx(hProcess, pbModule, inh64.OptionalHeader.SizeOfHeaders,
src/creatwth.cpp:     if (!VirtualProtectEx(hProcess, pbModule, inh64.OptionalHeader.SizeOfHeaders,
src/creatwth.cpp:         if (!DetourVirtualProtectSameExecuteEx(hProcess, der.pclr, sizeof(cldr), PAGE_READWRITE, &dwProtect)) {
src/creatwth.cpp:             DETOUR_TRACE(("VirtualProtectEx(cldr) write failed: %d\n", GetLastError()));
src/creatwth.cpp:             if (!VirtualProtectEx(hProcess, der.pclr, sizeof(cldr), dwProtect, &dwProtect)) {
src/creatwth.cpp:                 DETOUR_TRACE(("VirtualProtectEx(cldr) restore failed: %d\n", GetLastError()));
src/creatwth.cpp:     PBYTE pbBase = (PBYTE)VirtualAllocEx(hProcess, NULL, cbTotal,
src/creatwth.cpp:         DETOUR_TRACE(("VirtualAllocEx(%d) failed: %d\n", cbTotal, GetLastError()));
src/detours.cpp:     if (!VirtualProtect(pRegion, DETOUR_REGION_SIZE, PAGE_EXECUTE_READWRITE, &dwOld)) {
src/detours.cpp:         VirtualProtect(pRegion, DETOUR_REGION_SIZE, PAGE_EXECUTE_READ, &dwOld);
src/detours.cpp:         PVOID pv = virtualAlloc(pbTry,
src/detours.cpp:         PVOID pv = virtualAlloc(pbTry,
src/detours.cpp:         VirtualProtect(o->pbTarget, o->pTrampoline->cbRestore,
src/detours.cpp:         VirtualProtect(o->pbTarget, o->pTrampoline->cbRestore, o->dwPerm, &dwOld);
```

Don't Be a paster - wbenny DetoursNT

The screenshot shows the GitHub repository page for 'wbenny / DetoursNT'. At the top, it indicates the repository is 'Public' and shows 23 watches, 89 forks, and 377 stars. Below this are navigation tabs for Code, Issues (2), Pull requests, Actions, Projects, Wiki, Security, and Insights. The main content area shows the repository structure with a table of files and folders, and an 'About' section on the right.

File/Folder	Description	Commit Date
Detours @ c0c0ef9	initial commit	4 years ago
DetoursNT	retarget solution to latest MSVC	9 months ago
Images	initial commit	4 years ago
Sample	retarget solution to latest MSVC	9 months ago
SampleHookDLL	retarget solution to	
...	initial commit	

About
Detours with just single dependency - NTDLL

Tags: windows, detours, hooking, ntdll

Readme, MIT License, 377 stars

The screenshot shows a Windows File Explorer window titled 'Dependencies (x64)'. The address bar displays the path 'D:\Collections\University\FYP\sEDRDLL_V0.1\Bin\x64\Debug\SampleHookDLL.dll'. The main pane shows a list of files, with 'C:\Windows\system32\ntdll.dll' highlighted.

File View Options Help

SampleHookDLL

D:\Collections\University\FYP\sEDRDLL_V0.1\Bin\x64\Debug\SampleHookDLL.dll

C:\Windows\system32\ntdll.dll



Super easy hooking of NTAPI

```
typedef NTSTATUS(NTAPI* fnNtAllocateVirtualMemory)(
    HANDLE ProcessHandle,
    PVOID* BaseAddress,
    ULONG_PTR ZeroBits,
    PSIZE_T RegionSize,
    ULONG AllocationType,
    ULONG Protect
);

static fnNtAllocateVirtualMemory OrigNtAllocateVirtualMemory;

EXTERN_C NTSTATUS NTAPI HookNtAllocateVirtualMemory(
    HANDLE ProcessHandle,
    PVOID* BaseAddress,
    ULONG_PTR ZeroBits,
    PSIZE_T RegionSize,
    ULONG AllocationType,
    ULONG Protect
)
{
    DbgPrint("wedoalittlebitoftooling");
    return OrigNtAllocateVirtualMemory(ProcessHandle, BaseAddress, ZeroBits, RegionSize, AllocationType, Protect);
}
```



Still need to grab pointers to win32



Resolving function pointers without win32 in a reliable way.

```
/* [...SNIP...] */
RtlInitUnicodeString(&ModuleNameString_U, L"kernelbase");
Status = LdrLoadDll(UNICODE_NULL, NULL, &ModuleNameString_U, &ModuleHandle);
/* [...SNIP...] */
RtlInitString(&ProcedureNameString, "VirtualAlloc");
Status = LdrGetProcedureAddress(
    ModuleHandle,
    &ProcedureNameString,
    (ULONG) NULL,
    (PVOID*)&ProcedurePointer
);

/* [...SNIP...] */
/* have a pointer to the function */
void* VirtualAllocPointer = ProcedurePointer;
/* have a clean pointer to keep */
OrigVirtualAlloc = (fnVirtualAlloc)VirtualAllocPointer;
```



So now what?

- We know how to clean a PE and force it to only use NTDLL.DLL
- We have a hooking library which does most of the heavy lifting and solely depends on NTDLL.DLL
- We can hook NTDLL & Win32 functions

What now?

- Meaningful information for humans
- I did not understand why I would get recursive hook hits, hindsight is 20:20
- How did visual studio know what a pointer pointed to (And I wanted to not use pdb / symbols)
- Learning learning learning, if you aren't learning there isn't much point living.



Caller module name acquisition.



Getting the Calling module by name

PEB->InMemoryOrderModuleList = PEB_LDR_DATA Struct

“The head of a doubly-linked list that contains the loaded modules for the process. Each item in the list is a pointer to an LDR_DATA_TABLE_ENTRY structure. For more information, see Remarks.” - msdn , PEB_LDR_DATA

Good place to look / get info about processes. Can be requested from the kernel by asking for the teb and selecting the current PEB then copying out/referencing the struct.



We all know about the PEB


Process Environment Block

<https://github.com/processhacker/phnt/blob/master/ntpebteb.h#L69>

Nice

Basic PEB walk

- Get PEB -> InMemoryOrderModuleList -> Iterate until your pointer is in the DLL's memory range for that process



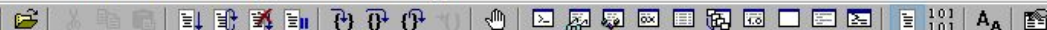
```
typedef struct _PEB_LDR_DATA {
    BYTE          Reserved1[8];
    PVOID         Reserved2[3];
    LIST_ENTRY    InMemoryOrderModuleList;
} PEB_LDR_DATA, *PPEB_LDR_DATA;

typedef struct _LIST_ENTRY {
    struct _LIST_ENTRY *Flink;
    struct _LIST_ENTRY *Blink;
} LIST_ENTRY, *PLIST_ENTRY, *RESTRICTED_POINTER
PRLIST_ENTRY;

typedef struct _LDR_DATA_TABLE_ENTRY {
    PVOID Reserved1[2];
    LIST_ENTRY InMemoryOrderLinks;
    PVOID Reserved2[2];
    PVOID DllBase;
    PVOID EntryPoint;
    PVOID Reserved3;
    UNICODE_STRING FullDllName;
    BYTE Reserved4[8];
    PVOID Reserved5[3];
    union {
        ULONG CheckSum;
        PVOID Reserved6;
    };
    ULONG TimeDateStamp;
} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;
```


C:\Windows\System32\calc.exe - WinDbg:10.0.22000.194 AMD64

File Edit View Debug Window Help



Command

```
0:000> !peb
PEB at 00000097e3674000
  InheritedAddressSpace: No
  ReadImageFileExecOptions: No
  BeingDebugged: Yes
  ImageBaseAddress: 00007ff73a7f0000
  NtGlobalFlag: 70
  NtGlobalFlag2: 0
  Ldr: 00007ffd9317a4c0
  Ldr.Initialized: Yes
  Ldr.InInitializationOrderModuleList: 0000025e78582ed0 . 0000025e785835f0
  Ldr.InLoadOrderModuleList: 0000025e78583080 . 0000025e785892a0
  Ldr.InMemoryOrderModuleList: 0000025e78583090 . 0000025e785892b0
    Base TimeStamp Module
    7ff73a7f0000 0340c410 Sep 24 17:02:24 1971 C:\Windows\System32\calc.exe
    7ffd93010000 1be73aa8 Nov 01 06:31:04 1984 C:\Windows\SYSTEM32\ntdll.dll
    7ffd92ad0000 f32175d9 Apr 05 08:12:25 2099 C:\Windows\System32\KERNEL32.DLL
    7ffd909c0000 0833f2d4 May 12 21:23:48 1974 C:\Windows\System32\KERNELBASE.dll
    7ffd91490000 a3fc69da Mar 07 22:06:18 2057 C:\Windows\System32\SHELL32.dll
    7ffd90f20000 39255ccf May 19 16:25:03 2000 C:\Windows\System32\msvcp_win.dll
    7ffd90730000 2bd748bf Apr 23 02:39:11 1993 C:\Windows\System32\ucrtbase.dll
    7ffd92880000 e1c9e1a1 Jan 14 19:08:17 2090 C:\Windows\System32\USER32.dll
    7ffd90990000 0dcd0213 May 03 21:26:59 1977 C:\Windows\System32\win32u.dll
    7ffd91070000 a0528517 Mar 27 12:16:23 2055 C:\Windows\System32\GDI32.dll
    7ffd90c90000 62042741 Feb 09 20:42:41 2022 C:\Windows\System32\gdi32full.dll
    7ffd91bf0000 564f9f39 Nov 20 22:31:21 2015 C:\Windows\System32\msvcrt.dll
    7ffd92a20000 120819e1 Aug 03 10:33:21 1979 C:\Windows\System32\ADVAPI32.dll
    7ffd911c0000 53d57421 Jul 27 22:50:25 2014 C:\Windows\System32\sechost.dll
    7ffd92e40000 e1d764f5 Jan 25 01:08:05 2090 C:\Windows\System32\RPCRT4.dll
  SubSystemData: 0000000000000000
  ProcessHeap: 0000025e78580000
  ProcessParameters: 0000025e785825f0
  CurrentDirectory: 'C:\Program Files (x86)\Windows Kits\10\Debuggers\'
  WindowTitle: 'C:\Windows\System32\calc.exe'
  ImageFile: 'C:\Windows\System32\calc.exe'
  CommandLine: 'C:\Windows\System32\calc.exe'
  DllPath: '< Name not readable >'
  Environment: 0000025e78581130
```



EH record information gathering.



_ReturnAddress() MSVC compiler intrin

```
#include <stdio.h>
#include <intrin.h>
#pragma intrinsic(_ReturnAddress)

void returnval(void)
{
    printf("Return Address 0x%p\n", (void *)_ReturnAddress());
    /* this will point to the line after the function call */
}

void main(void)
{
    returnval();
    /* <- Here is where the returnvalue will logically point to */
}
```



- File: KernelBase.dll
 - Dos Header
 - Nt Headers
 - File Header
 - Optional Header
 - Data Directories [x]
 - Section Headers [x]
 - Export Directory
 - Import Directory
 - Resource Directory
 - Exception Directory
 - Relocation Directory
 - Debug Directory
 - Address Converter
 - Dependency Walker
 - Hex Editor
 - Identifier
 - Import Adder
 - Quick Disassembler
 - Rebuilder
 - Resource Editor

KernelBase.dll

Name	Virtual Size	Virtual Addr...	Raw Size	Raw Address	Reloc Address	Linenum
00000270	00000278	0000027C	00000280	00000284	00000288	0000028
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword
.text	0011086D	00001000	00110A00	00000400	00000000	0000000
.rdata	00177AD4	00112000	00177C00	00110E00	00000000	0000000
.data	00004440	0028A000	00001400	00288A00	00000000	0000000
.pdata	0000E748	0028F000	0000E800	00289E00	00000000	0000000
.didat	000006C8	0029E000	00000800	00298600	00000000	0000000
.nsr	00000548	0029F000	00000600	00298E00	00000000	0000000

This section contains:

Exception Directory: 0028F000



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	10	10	00	00	B5	10	00	00	80	1B	26	00	C0	10	00	00	00 .. μ0 .. 10 & . Å0 ..
00000010	11	11	00	00	98	1B	26	00	18	11	00	00	89	11	00	00	00 .. 10 & .00 .. 10 ..
00000020	A8	1B	26	00	90	11	00	00	36	14	00	00	8C	1F	26	00	'0 & . 0 .. 60 .. 1 & .
00000030	3C	14	00	00	FD	15	00	00	BC	1F	26	00	04	16	00	00	<0 .. 90 .. 34 & .00 ..
00000040	53	16	00	00	E8	1F	26	00	90	16	00	00	03	17	00	00	50 .. è & . 0 .. 00 ..
00000050	00	20	26	00	0C	17	00	00	99	17	00	00	18	20	26	00	.. & . 10 .. 10 .. 0 & .
00000060	A0	17	00	00	3A	18	00	00	30	20	26	00	40	18	00	00	0 .. :0 .. 0 & .@0 ..



RtlLookupFunctionEntry

```
typedef struct _IMAGE_RUNTIME_FUNCTION_ENTRY {
    DWORD BeginAddress;
    DWORD EndAddress;
    union {
        DWORD UnwindInfoAddress;
        DWORD UnwindData;
    } DUMMYUNIONNAME;
} RUNTIME_FUNCTION, *PRUNTIME_FUNCTION, _IMAGE_RUNTIME_FUNCTION_ENTRY, *_PIMAGE_RUNTIME_FUNCTION_ENTRY;

EXTERN_C NTSYSAPI PRUNTIME_FUNCTION NTAPI RtlLookupFunctionEntry(
    /* [in] */ DWORD64 ControlPc,
    /* [out] */ PDWORD64 ImageBase,
    /* [out] */ PUNWIND_HISTORY_TABLE HistoryTable
);
[...SNIP...]
runfunc = RtlLookupFunctionEntry(u1RetAddr, &imgbase, &HistTable);
```

Intellisense

```
51  EXTERN_C NTSTATUS NTAPI HookNtAllocateVirtualMemory(  
52      HANDLE      ProcessHandle,  
53      PVOID* BaseAddress,  
54      ULONG_PTR   ZeroBits,  
55      PSIZE_T     RegionSize,  
56      ULONG       AllocationType,  
57      ULONG       Protect  
58  )  
59  {  
60      void* returnaddress = _ReturnAddress();  
61      DbgPrint("HookNtAllocateVirtualMemory 0x%p\n", _ReturnAddress()); ≤1ms elapsed  
62      return OrigNtAllocateVirtualMemory(ProcessHandle,BaseAddress,ZeroBits,RegionSize,AllocationType,Protect);  
63  }
```

1

returnaddress 0x00007ff98ea318e8 (Inside KernelBase.dll!VirtualAlloc(void))

Caveats

/* HOOK Function */

```
EXTERN_C LPVOID WINAPI HookVirtualAlloc(  
    /*[in, optional]*/ LPVOID lpAddress,  
    /*[in]           */ SIZE_T dwSize,  
    /*[in]           */ DWORD  flAllocationType,  
    /*[in]           */ DWORD  flProtect  
);
```

```
//DbgPrint("HookVirtualAlloc 0x%p\n", _ReturnAddress());
```

```
void* returnaddress = ReturnAddress();
```

```
DbgPrint("H returnaddress 0x00007ff6620c18c9 {Sample.exe!main(void). Line 8} ↵
```

```
return OrigVirtualAlloc(lpAddress, dwSize, flAllocationType, flProtect);
```

```
332 EXTERN_C NTSTATUS WINAPI HookNtAllocateVirtualMemory( 56  
333     HANDLE ProcessHandle, 57  
334     PVOID* BaseAddress, 58  
335     ULONG_PTR ZeroBits, 59  
336     PSIZE_T RegionSize, 60  
337     ULONG AllocationType, 61  
338     ULONG Protect 62  
339 ) 63  
340 { 64  
341     DbgPrint("NTAPI:HookNtAllocateVirtualMemory\n"); 65  
342     retaddr 0x00007ff66d7e715 {Inside ntdll.dll!RtlpFindAndCommitPages(void)} 66  
343     void* retaddr = _ReturnAddress(); 67  
344     const WCHAR* Result = ModuleHunter(retaddr); ≤ 1ms elapsed 68  
345     WCHAR LineToLog[_MAX_PATH]; // e.g. kernelbase.dll 69  
346 70  
347     _snwprintf(LineToLog, _MAX_PATH, L"%ws", Result); 71  
348     LogLine("NTAPI:HookNtAllocateVirtualMemory", LineToLog, 1 72  
349 73  
74
```



Implementing the technique



We can clean out DLL from deps ready to be turned into pic / be early loaded by kapc

We can obtain the module name of a caller which hits any hook

We can obtain the name of a function calling our hook

We can trace a call to VirtualAlloc all the way down to the kernel, intercepting parameters on each sub api



Now it's just a case of putting the pieces together

Within a hook:

- Use `ReturnAddress()` compiler intrinsic to grab the RIP/PC
- Call `RtlLookupFunctionEntry` passing the RIP which returns a `RUNTIME_FUNCTION`
- We now have a pointer to the function which called the current hook
- We can get a copy of the PEB and Iterate the doubly linked list `InMemoryOrderModuleList` to get the caller module name
- PE parse the caller module and get the `IMAGE_EXPORT_DIRECTORY`
- Use the `IMAGE_EXPORT_DIRECTORY->NumberOfNames` to parse the PE and compare the pointer to the calling function to the current item in the iteration
- done, we now have the win32/NTAPI module in plaintext and the function name in plaintext

Humans be dumb

```

void WhoCalled(void* PCValue)
{
    ULONGLONG imgbase;
    ULONGLONG ullRetAddr          = (ULONGLONG)PCValue;
    PRUNTIME_FUNCTION runfunc     = NULL;
    UNWIND_HISTORY_TABLE HistTable = { 0 };

    /* search for RUNTIME_FUNCTION entry which describes current function */
    /* NOTE: Use the ntdll.dll export instead of win32 as win32 doesn't pop HistTable for some reason, very weird */
    runfunc = RtlLookupFunctionEntry(ullRetAddr, &imgbase, &HistTable);

    if (runfunc)
    {
        /*
        If a function table entry is found, RIP can lie within three regions:
        a) in an epilog,
        b) in the prolog,
        c) in code that may be covered by an exception handler.
        */

        void* addToSearch = (void*)(imgbase + (ULONGLONG)(runfunc->BeginAddress));
        PPEB pPEB = (PPEB)__readgsqword(0x60);
        PEB_LDR_DATA* peb_ldr_data = pPEB->Ldr;
        LIST_ENTRY* list_head = &(peb_ldr_data->InMemoryOrderModuleList);
        LIST_ENTRY* list_entry;
        LDR_DATA_TABLE_ENTRY_COMPLETED* ldr_entry;

        /* for items in InMemoryOrderModuleList do; */
        for (list_entry = list_head->Flink; list_entry != list_head; list_entry = list_entry->Flink)
        {
            /* We follow inMemoryOrder, so list_entry points to LDR_DATA_TABLE_ENTRY_COMPLETED.InMemoryOrderLinks */
            /* We need to remove the size of the first element to get the address of the object */
            ldr_entry = (LDR_DATA_TABLE_ENTRY_COMPLETED*)((char*)list_entry - sizeof(LIST_ENTRY));
            void * totSize = (UINT64*)ldr_entry->DllBase + (UINT64)ldr_entry->SizeOfImage;

            /* check if the PC/RIP is within the current DLL taken from current ldr_entry (totSize = base + size of image) */
            if (PCValue > *(UINT64*)(void*)ldr_entry->DllBase && PCValue < totSize)
            {
                DbgPrint("%ws called this function\n", ldr_entry->BaseDllName.Buffer);
                /* Normal iredteam type PE parse for function */
            }
        }
    }
}

```



Demo



Humans like words

HOOKHIT: NTAPI:HookNtAllocateVirtualMemory,	(caller module: NOT_OG:ntdll.dll:NonExportedFunc),	(count : 1), (func :0x00007FFA33F7E715)
HOOKHIT: WIN32:HookVirtualAlloc, HOOKHIT: NTAPI:HookNtAllocateVirtualMemory,	(caller module: NOT_OG:malproc6464d.exe:UnknownCaller), (caller module: NOT_OG:KERNELBASE.dll:VirtualAlloc),	(count : 1), (func :0x00007FF7DB1319A2) (count : 1), (func :0x00007FFA31921998)
HOOKHIT: WIN32:HookVirtualAllocEx, HOOKHIT: WIN32:HookVirtualAllocExNuma, HOOKHIT: NTAPI:HookNtAllocateVirtualMemory,	(caller module: NOT_OG:malproc6464d.exe:UnknownCaller), (caller module: NOT_OG:KERNELBASE.dll:VirtualAllocEx), (caller module: NOT_OG:KERNELBASE.dll:VirtualAllocExNuma),	(count : 1), (func :0x00007FF7DB1319C9) (count : 1), (func :0x00007FFA31934C16) (count : 1), (func :0x00007FFA31934C7D)
HOOKHIT: WIN32:HookVirtualAllocExNuma, HOOKHIT: NTAPI:HookNtAllocateVirtualMemory, HOOKHIT: WIN32:HookCreateRemoteThreadEx, HOOKHIT: NTAPI:HookNtCreateThreadEx, HOOKHIT: NTAPI:HookNtAllocateVirtualMemory,	(caller module: NOT_OG:malproc6464d.exe:UnknownCaller), (caller module: NOT_OG:KERNELBASE.dll:VirtualAllocExNuma), (caller module: NOT_OG:KERNEL32.DLL:CreateThread), (caller module: NOT_OG:KERNELBASE.dll:CreateRemoteThreadEx), (caller module: NOT_OG:ntdll.dll:NonExportedFunc),	(count : 1), (func :0x00007FF7DB1319F8) (count : 1), (func :0x00007FFA31934C7D) (count : 1), (func :0x00007FFA33BCB5DD) (count : 1), (func :0x00007FFA318F55EF) (count : 1), (func :0x00007FFA33F7E715)
HOOKHIT: NTAPI:HookNtAllocateVirtualMemory, HOOKHIT: NTAPI:HookNtAllocateVirtualMemory,	(caller module: NOT_OG:ntdll.dll:RtlCreateHeap), (caller module: NOT_OG:ntdll.dll:RtlCreateHeap),	(count : 1), (func :0x00007FFA33F7AC40) (count : 1), (func :0x00007FFA33F7ACF2)
HOOKHIT: NTAPI:HookNtAllocateVirtualMemory, HOOKHIT: NTAPI:HookNtAllocateVirtualMemory, HOOKHIT: NTAPI:HookNtAllocateVirtualMemory, HOOKHIT: NTAPI:HookNtAllocateVirtualMemory, HOOKHIT: NTAPI:HookNtAllocateVirtualMemory, HOOKHIT: NTAPI:HookNtAllocateVirtualMemory, HOOKHIT: NTAPI:HookNtAllocateVirtualMemory,	(caller module: NOT_OG:ntdll.dll:NonExportedFunc), (caller module: NOT_OG:ntdll.dll:NonExportedFunc), (caller module: NOT_OG:ntdll.dll:NonExportedFunc), (caller module: NOT_OG:ntdll.dll:NonExportedFunc), (caller module: NOT_OG:ntdll.dll:NonExportedFunc), (caller module: NOT_OG:ntdll.dll:NonExportedFunc), (caller module: NOT_OG:ntdll.dll:NonExportedFunc),	(count : 1), (func :0x00007FFA33F7E715) (count : 1), (func :0x00007FFA33F7E715) (count : 1), (func :0x00007FFA33F7E715) (count : 1), (func :0x00007FFA33FB489A) (count : 1), (func :0x00007FFA33FB48F1) (count : 1), (func :0x00007FFA33F7E715) (count : 1), (func :0x00007FFA33F7E715)



Dtrace for windows

Dtrace was a project introduced in solaris 10 but has been brought out for windows recently

Provides function boundary tracing and syscall hooking on entry and return all from the kernel with a usermode d programming interface.

traceext.sys acts as a bridge between ntoskrnl and the dtrace implementation and allows for simple tracing of syscalls and etw

Syscall tracing -> callback on each syscall within the kernel and allows for tracing without affecting usermode components



Demo 2

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

void main(void)
{
    printf("start\n");
    getchar();
    void* exec_mem = NULL;
    exec_mem = VirtualAlloc(0, 4096, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
    printf("end\n");
    getchar();
}
```



Thanks

Shoutouts/references:

wbenny

- <https://github.com/wbenny>

phnt lib

- <https://github.com/processhacker/phnt>

Lucasg

- <https://lucasg.github.io/Dependencies/>

namazso/Mattiwatti/jonaslyk/Daax

- <https://secret.club/>

dennisbabkin

- <https://github.com/dennisbabkin>

Rbmm

- <https://github.com/rbmm>

jonaslyk

- mentor <https://twitter.com/jonaslyk>



When I have some time

Working on a couple of side projects

- Different take on a driver LPE which I can't seem to have been done before
- writing a stage 1 implant / c2

@PUNICODE_STRING






Le fin.







UNUSED SLIDES

- 
- You don't really care
 - But I if you do;
 - Some certs, OSCP, CRTO, CPSA, s7_RTO_MDE, s7_RTO_MDI (all certs no skill)
 - Fresh out of university, 1st class in Cyber Security with Digital Forensics.
 - I like HTB prolabs and Windows network tomfoolery.
 - Security consultant with a nice new company.

Shoutouts/references:

wbenny

- <https://github.com/wbenny>

phnt lib

- <https://github.com/processhacker/phnt>

Lucasg

- <https://lucasg.github.io/Dependencies/>

namazso/Mattiwatti/jonaslyk/Daax

- <https://secret.club/>

dennisbabkin

- <https://github.com/dennisbabkin>

Rbmm

- <https://github.com/rbmm>

dad98 (@rad9800)

- <https://twitter.com/rad9800>



Why / What removal of CRT + Win32 strip?

- 65192 bytes -> 24576 bytes.
- .00cfg
- .pdata
- Depend on yourself (Ability to early load)