

Forum:[Hack Forums](#)

Thread: [C++ error when manually mapping NTDLL into current process](#)

redbaron - May 3, 2024, 2:43 pm

i am attempting to write some c++ code that manually loads a fresh copy of ntdll from disk into memory. heres how it works:

- walk PEB to get ntdll base address and path to file on disk
- dynamically load functions from this ntdll to be used
- call ntcreatefile and ntreadfile to load the file from disk into memory
- call ntcreatesection and ntmapviewofsection twice to create 2 new memory spaces, one with RW and the other with RX permissions
- copy headers + data in each section header into section with RW permissions
- finally load functions dynamically from this new section by comparing export names with hashes

however when i call any function loaded from this new ntdll i get the exact same error: "Access violation reading location 0x4B3AB370."

When i check the disassembly this address is just empty

can someone explain to me whats going on?

JohnFitzgeraldKennedy - May 3, 2024, 3:50 pm

Can you post code snippets of the export function address lookup code and also the DLL mapping/copying of section information?

It may also be useful to attach a debugger and set a breakpoint before the exported function actually gets called to double check the disassembly at the address you get from ntdll exports. If that adresss dissembly isn't right im guessing you have a bug somewhere in the above mentioned parts of the logic.

Hackers - May 4, 2024, 6:42 am

Are you 100% sure you are remapping the base address correctly?

Look into how to reset the base, how to rewrite import tables if needed, and related changes.

You can look into what I did with Bubble Crypt to see some of this in action.

COCKBOT - May 4, 2024, 7:37 am

(May 4, 2024 06:42 AM)nodes Wrote: [\[CITE\]](#) Are you 100% sure you are remapping the base address correctly?

Look into how to reset the base, how to rewrite import tables if needed, and related changes.

You can look into what I did with Bubble Crypt to see some of this in action.

This.

For all we know, you've already confirmed that each step of your process up to this point is working as intended (each step, not all steps together working, obviously).

But it could also be that there's a margin for error stemming from assumptions you made.

Step through your code line by line and confirm that everything that is currently happening is correct (aside from the fact that you aren't able to map the module's contents).

I don't normally preach like this, I typically find it annoying. But the amount of times I've hit a wall when working with Windows internals for hours only to realize the entirety of my problems lie within some seemingly-benign assumption I made hours prior, is alarmingly high.

Edit: Remember to strip the pe header from the module after you map it.

redbaron - May 4, 2024, 8:49 am

(May 4, 2024 07:37 AM)COCKBOT Wrote: [\[CITE\]](#) Edit: Remember to strip the pe header from the module after you map it.

Could you elaborate on how to do this please?

(May 3, 2024 03:50 PM)JohnFitzgeraldKennedy Wrote: [\[CITE\]](#) Can you post code snippets of the export function address lookup code and also the DLL mapping/copying of section information?

It may also be useful to attach a debugger and set a breakpoint before the exported function actually gets called to double check the disassembly at the address you get from ntdll exports. If that adresss dissembly isn't right im guessing you have a bug somewhere in the above mentioned parts of the logic.

Here is the code

when i call any of the functions loaded i get the error i mentionated in the first post

if i replace "sectionMapping_RX" with the base address of ntdll from the PEB when loading any of the functions they work fine when called

COCKBOT - May 5, 2024, 12:15 am

(May 4, 2024 08:49 AM)SIM-5073947 Wrote: [\[CITE\]](#) Could you elaborate on how to do this please?

Here is the code

-snip-

when i call any of the functions loaded i get the error i mentionated in the first post

if i replace "sectionMapping_RX" with the base address of ntdll from the PEB when loading any of the functions they work fine when called

Here's your code, but with proper syntax highlighting: <https://p.ryswick.net/p/ONVcOO.cpp>

As far as stripping the PE header goes, look at the GH injector.

redbaron - May 5, 2024, 4:20 am

(May 5, 2024 12:15 AM)COCKBOT Wrote: [\[CITE\]](#) As far as stripping the PE header goes, look at the GH injector.

Will do, thanks for the help

COCKBOT - May 5, 2024, 6:30 am

(May 5, 2024 04:20 AM)SIM-5073947 Wrote: [\[CITE\]](#) Will do, thanks for the help

Np men. Glhf.

Finacustech - May 6, 2024, 4:08 am

The "Access violation reading location" error typically indicates that your program is attempting to access memory that it does not have permission to access. This can happen for a variety of reasons, including:

Memory Protection: The memory you are trying to access may be protected, either by the operating system or by your program itself. This could be due to the memory being marked as read-only (RX) or no-access.

Invalid Memory Address: The memory address you are attempting to access may be invalid or uninitialized. This could happen if the memory was not properly allocated or if it was deallocated before your program attempted to access it.

Memory Corruption: There may be a bug in your code that is corrupting memory, causing the access violation error when you attempt to access it.

Address Space Layout Randomization (ASLR): ASLR randomizes the memory addresses of loaded modules, including DLLs like ntdll.dll. If you are attempting to manually load ntdll.dll into memory, you need to ensure that you are resolving function addresses correctly, taking into account any address randomization that may have occurred.

Given your description, it's likely that the issue lies with either memory protection or address resolution. Make sure that you are properly allocating and protecting the memory regions you are using, and ensure that you are resolving function addresses correctly after manually loading ntdll.dll into memory.

It's also worth mentioning that manually loading system DLLs like ntdll.dll is a complex and error-prone process, and can lead to stability and security issues if not done correctly. Consider whether there are alternative approaches to achieve your goal that do not involve manually loading system DLLs.

redbaron - May 21, 2024, 1:07 pm

(May 6, 2024 04:08 AM)Finacustech Wrote: [\[CITE\]](#) The "Access violation reading location" error typically indicates that your program is attempting to access memory that it does not have permission to access. This can happen for a variety of reasons, including:

Memory Protection: The memory you are trying to access may be protected, either by the operating system or by your program itself. This could be due to the memory being marked as read-only (RX) or no-access.

Invalid Memory Address: The memory address you are attempting to access may be invalid or uninitialized. This could happen if the memory was not properly allocated or if it was deallocated before your program attempted to access it.

Memory Corruption: There may be a bug in your code that is corrupting memory, causing the access violation error when you attempt to access it.

Address Space Layout Randomization (ASLR): ASLR randomizes the memory addresses of loaded modules, including DLLs like ntdll.dll. If you are attempting to manually load ntdll.dll into memory, you need to ensure that you are resolving function addresses correctly, taking into account any address randomization that may have occurred.

Given your description, it's likely that the issue lies with either memory protection or address resolution. Make sure that you are properly allocating and protecting the memory regions you are using, and ensure that you are resolving function addresses correctly after manually loading ntdll.dll into memory.

It's also worth mentioning that manually loading system DLLs like ntdll.dll is a complex and error-prone process, and can lead to stability and security issues if not done correctly. Consider

whether there are alternative approaches to achieve your goal that do not involve manually loading system DLLs.

chatgpt goes hard

RedNode - May 21, 2024, 2:35 pm

the problem with memory please include the code to check it

redbaron - May 24, 2024, 6:29 am

(May 21, 2024 02:35 PM)RedNode Wrote: [\[CITE\]](#) the problem with memory please include the code to check it

this is all the code there is

Hackers - May 26, 2024, 7:10 am

(May 24, 2024 06:29 AM)pooplicker Wrote: [\[CITE\]](#) this is all the code there is

it's because you don't have permission to access the memory. you need to handle the new base address & all related relocations.

here is a relevant threads of mine:

<https://hackforums.net/showthread.php?tid=6266052>

I do some of this relocation work in bubble pop:

<https://hackforums.net/showthread.php?tid=6266614>

You have started the process, but you don't have everything done yet for the address mapping. Check out the following snippets from bubble pop :)

```
Code char *mem = NULL;
mem = (char *) malloc(100000000);
if (mem != NULL) {
    memset(mem, 00, 100000000);
    free(mem);
    PIMAGE_DOS_HEADER dosHeader = (PIMAGE_DOS_HEADER)programData;
    PIMAGE_NT_HEADERS ntHeaders = (PIMAGE_NT_HEADERS)(programData +
dosHeader->e_lfanew);
    MemoryManager memMgr;
    PVOID baseAddress =
memMgr.AllocateMemory((void*)ntHeaders->OptionalHeader.ImageBase,
ntHeaders->OptionalHeader.SizeOfImage, PAGE_READWRITE);
    if (!baseAddress) {
        baseAddress = memMgr.AllocateMemory(nullptr,
ntHeaders->OptionalHeader.SizeOfImage, PAGE_READWRITE);
        if (!baseAddress) {
            return 1;
        }
    }
    memcpy(baseAddress, programData, ntHeaders->OptionalHeader.SizeOfHeaders);
    PIMAGE_SECTION_HEADER sectionHeader = IMAGE_FIRST_SECTION(ntHeaders);
    for (WORD i = 0; i < ntHeaders->FileHeader.NumberOfSections; ++i) {
        PVOID sectionDestination = (PBYTE)baseAddress + sectionHeader[i].VirtualAddress;
        memcpy(sectionDestination, programData + sectionHeader[i].PointerToRawData,
sectionHeader[i].SizeOfRawData);
    }
}
```

```

    if (baseAddress != (PVOID)ntHeaders->OptionalHeader.ImageBase) {
        PIMAGE_BASE_RELOCATION relocation =
(PIMAGE_BASE_RELOCATION)((LPBYTE)baseAddress +
ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_BASERELOC].VirtualAd
dress);
        while (relocation->VirtualAddress) {
            if (relocation->SizeOfBlock >= sizeof(IMAGE_BASE_RELOCATION)) {
                int count = (relocation->SizeOfBlock - sizeof(IMAGE_BASE_RELOCATION)) /
sizeof(WORD);
                WORD* relocItem = (WORD*)((LPBYTE)relocation +
sizeof(IMAGE_BASE_RELOCATION));
                for (int i = 0; i < count; i++) {
                    if (relocItem[i] >> 12 == IMAGE_REL_BASED_DIR64) {
                        ULONGLONG* patchAddrHL = (ULONGLONG*)((LPBYTE)baseAddress +
relocation->VirtualAddress + (relocItem[i] & 0xFFF));
                        *patchAddrHL += ((ULONGLONG)baseAddress -
ntHeaders->OptionalHeader.ImageBase);
                    }
                }
            }
            relocation = (PIMAGE_BASE_RELOCATION)((LPBYTE)relocation +
relocation->SizeOfBlock);
        }
    }
    ProcessImportTable(baseAddress);
    LPVOID entryPoint = (PBYTE)baseAddress +
ntHeaders->OptionalHeader.AddressOfEntryPoint;
    memMgr.RevertCodeSectionProtection(baseAddress,
ntHeaders->OptionalHeader.SizeOfImage, PAGE_EXECUTE_READWRITE);
    ((void (*)(void))entryPoint)();
    memMgr.RevertCodeSectionProtection(baseAddress,
ntHeaders->OptionalHeader.SizeOfImage, PAGE_READWRITE);
}
another relevant func:
Code void ProcessImportTable(PVOID baseAddress) {
    PIMAGE_DOS_HEADER dosHeader = (PIMAGE_DOS_HEADER)baseAddress;
    PIMAGE_NT_HEADERS ntHeaders = (PIMAGE_NT_HEADERS)((LPBYTE)baseAddress +
dosHeader->e_lfanew);
    PIMAGE_IMPORT_DESCRIPTOR importDesc =
(PIMAGE_IMPORT_DESCRIPTOR)((LPBYTE)baseAddress +
ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_IMPORT].VirtualAddress
);

    while (importDesc->Name) {
        LPCSTR dllName = (LPCSTR)((LPBYTE)baseAddress + importDesc->Name);
        HMODULE hModule = LoadLibraryA(dllName);

        if (!hModule) {
            // printf("Failed to load %s\n", dllName);
            importDesc++;
            continue;
        }
    }
}

```

```

    PIMAGE_THUNK_DATA thunk = (PIMAGE_THUNK_DATA)((LPBYTE)baseAddress +
importDesc->FirstThunk);
    while (thunk->u1.AddressOfData) {
        FARPROC pFunc = NULL;
        if (thunk->u1.Ordinal & IMAGE_ORDINAL_FLAG) {
            pFunc = GetProcAddress(hModule, (LPCSTR)IMAGE_ORDINAL(thunk->u1.Ordinal));
        } else {
            PIMAGE_IMPORT_BY_NAME importByName =
(PIMAGE_IMPORT_BY_NAME)((LPBYTE)baseAddress + thunk->u1.AddressOfData);
            pFunc = GetProcAddress(hModule, (LPCSTR)&importByName->Name);
        }

        if (pFunc) {
            *(FARPROC*)thunk = pFunc;
        }
        thunk++;
    }

    importDesc++;
}
}

```

Please also understand you can't just copy this code and use it, you'll need to work it in to your own program.

Here's the MemoryManager file from Bubble Pop:

Code #include "MemoryManager.h"

#include <iostream>

#include "skCrypter.h"

```

typedef NTSTATUS (NTAPI *pNtQueryInformationProcess)(
    HANDLE ProcessHandle,
    PROCESSINFOCLASS ProcessInformationClass,
    PVOID ProcessInformation,
    ULONG ProcessInformationLength,
    PULONG ReturnLength);

```

```

extern "C" NTSTATUS NTAPI NtQueryInformationProcess(
    HANDLE ProcessHandle,
    PROCESSINFOCLASS ProcessInformationClass,
    PVOID ProcessInformation,
    ULONG ProcessInformationLength,
    PULONG ReturnLength);

```

// Function to parse the export table and get function addresses

```

void* GetFunctionAddress(PVOID baseAddress, const char* functionName) {
    auto dosHeader = (PIMAGE_DOS_HEADER)baseAddress;
    auto ntHeaders = (PIMAGE_NT_HEADERS)((PBYTE)baseAddress + dosHeader->e_lfanew);

    auto exportDirectory = (PIMAGE_EXPORT_DIRECTORY)((PBYTE)baseAddress +
ntHeaders->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);

    auto functions = (PDWORD)((PBYTE)baseAddress + exportDirectory->AddressOfFunctions);
    auto names = (PDWORD)((PBYTE)baseAddress + exportDirectory->AddressOfNames);
    auto ordinals = (PWORD)((PBYTE)baseAddress + exportDirectory->AddressOfNameOrdinals);
}

```

```

for (DWORD i = 0; i < exportDirectory->NumberOfNames; i++) {
    const char* exportName = (const char*)baseAddress + names[i];
    if (_stricmp(exportName, functionName) == 0) {
        DWORD functionRVA = functions[ordinals[i]];
        return (PBYTE)baseAddress + functionRVA;
    }
}
return nullptr;
}

```

```

// Helper function to read a UNICODE_STRING from memory
std::wstring ReadUnicodeString(const UNICODE_STRING& unicodeString) {
    return std::wstring(unicodeString.Buffer, unicodeString.Length / sizeof(WCHAR));
}

```

```

MemoryManager::MemoryManager() {
    HMODULE ntdll = LoadLibraryW(skCrypt(L"ntdll.dll"));
    if (!ntdll) {
        // std::cerr << "Failed to load ntdll.dll." << std::endl;
        return;
    }
}

```

```

pNtQueryInformationProcess NtQueryInformationProcess =
(pNtQueryInformationProcess)GetProcAddress(ntdll, skCrypt("NtQueryInformationProcess"));
if (!NtQueryInformationProcess) {
    // std::cerr << "Failed to get NtQueryInformationProcess address." << std::endl;
    return;
}

```

```

HANDLE processHandle = GetCurrentProcess();
PROCESS_BASIC_INFORMATION pbi;
ZeroMemory(&pbi, sizeof(pbi));

```

```

NTSTATUS status = NtQueryInformationProcess(processHandle, ProcessBasicInformation,
&pbi, sizeof(pbi), nullptr);
if (status != 0) {
    // std::cerr << "Failed to get process information." << std::endl;
    return;
}

```

```

PEB* peb = pbi.PebBaseAddress;
PPEB_LDR_DATA ldr = peb->Ldr;
LIST_ENTRY* moduleList = &ldr->InMemoryOrderModuleList;

```

```

for (LIST_ENTRY* entry = moduleList->Flink; entry != moduleList; entry = entry->Flink) {
    LDR_DATA_TABLE_ENTRY* moduleEntry = CONTAINING_RECORD(entry,
LDR_DATA_TABLE_ENTRY, InMemoryOrderLinks);
    std::wstring dllName = ReadUnicodeString(moduleEntry->FullDllName);
}

```

```

if (dllName.find(skCrypt(L"ntdll.dll")) != std::wstring::npos) {
    auto dllBase = moduleEntry->DllBase;
    pNtAllocateVirtualMemory =

```

```

reinterpret_cast<PFN_NtAllocateVirtualMemory>(GetFunctionAddress(dllBase,
skCrypt("NtAllocateVirtualMemory"))));
    pNtFreeVirtualMemory =
reinterpret_cast<PFN_NtFreeVirtualMemory>(GetFunctionAddress(dllBase,
skCrypt("NtFreeVirtualMemory"))));
    if (!pNtAllocateVirtualMemory || !pNtFreeVirtualMemory) {
        // std::cerr << "Failed to get addresses of Nt functions." << std::endl;
        return;
    }
    break;
}
}
FreeLibrary(ntdll);
}

```

```

void* MemoryManager::AllocateMemory(void* preferredBaseAddress, SIZE_T size, DWORD
initialProtection) {
    if (!pNtAllocateVirtualMemory) {
        // std::cerr << skCrypt("NtAllocateVirtualMemory function not loaded.") << std::endl;
        return nullptr;
    }

    PVOID baseAddress = preferredBaseAddress;
    SIZE_T regionSize = size;
    NTSTATUS status = pNtAllocateVirtualMemory(
        GetCurrentProcess(),
        &baseAddress,
        0,
        &regionSize,
        MEM_COMMIT | MEM_RESERVE,
        initialProtection); // Use the initialProtection parameter

    if (status != STATUS_SUCCESS) {
        // std::cerr << skCrypt("Failed to allocate memory.") << std::endl;
        return nullptr;
    }

    return baseAddress;
}

```

```

void MemoryManager::RevertCodeSectionProtection(void* baseAddress, SIZE_T size, DWORD
initialProtection) {
    DWORD oldProtection;
    if (!VirtualProtect(baseAddress, size, initialProtection, &oldProtection)) {
        // std::cerr << skCrypt("Failed to revert code section protection.") << std::endl;
        // Handle error
    }
}

```

```

bool MemoryManager::FreeMemory(void* address) {

```

```

if (!pNtFreeVirtualMemory) {
    // std::cerr << skCrypt("NtFreeVirtualMemory function not loaded.") << std::endl;
    return false;
}

PVOID baseAddress = address;
SIZE_T regionSize = 0; // Size is ignored with MEM_RELEASE
NTSTATUS status = pNtFreeVirtualMemory(
    GetCurrentProcess(),
    &baseAddress,
    &regionSize,
    MEM_RELEASE);

return status == STATUS_SUCCESS;
}

```

secret_code - June 1, 2024, 4:38 am

(May 4, 2024 08:49 AM)pooplicker Wrote: [\[CITE\]](#) Could you elaborate on how to do this please?

Here is the code

when i call any of the functions loaded i get the error i mentionated in the first post

if i replace "sectionMapping_RX" with the base address of ntdll from the PEB when loading any of the functions they work fine when called

hi there... some tips 4 your code: try to avoid using reinterpret_cast if possible, use ++x instead of x++, avoid using copy assignment initialization when possible, use direct initialization instead and use more robust error handling...

HFDone - June 3, 2024, 10:53 am

didnt see any error handling in your code do you understand what you are doing

0xDEAD10CC - June 12, 2024, 1:22 pm

(Jun 1, 2024 04:38 AM)secret_code Wrote: [\[CITE\]](#) hi there... some tips 4 your code: try to avoid using reinterpret_cast if possible, use ++x instead of x++, avoid using copy assignment initialization when possible, use direct initialization instead and use more robust error handling...

x++ vs ++x doesn't really matter as much because of other optimizations that can take place with the temporary. We no longer live in the 80s with archaic CPU and compiler optimizations, and we're contextually speaking about integers here, not even overloaded types where the difference could possibly be significant.

secret_code - June 14, 2024, 4:56 pm

(Jun 12, 2024 01:22 PM)0xDEAD10CC Wrote: [\[CITE\]](#) x++ vs ++x doesn't really matter as much because of other optimizations that can take place with the temporary. We no longer live in the 80s with archaic CPU and compiler optimizations, and we're contextually speaking about integers here, not even overloaded types where the difference could possibly be significant.

you better get used to use pre-increment instead post-increment my friend, you can be alright using post-increment for primitive types, but if you don't get used to apply pre-.....in your code you will eventually use post-increment with user-defined types as well which will cause performance "decrementing" in your code... if you want you can also use +x+, i can't care less, if you want to use

LQ code do it....