Reversing and Calling Windows 11 Native API Syscalls with x64 Assembly

(An oldy but a goody)

Daniel O'Malley

Why Do This?

Sure, we COULD use the normal/recommended Win API in Kernel32.dll (either via C++ or Assembly), but…

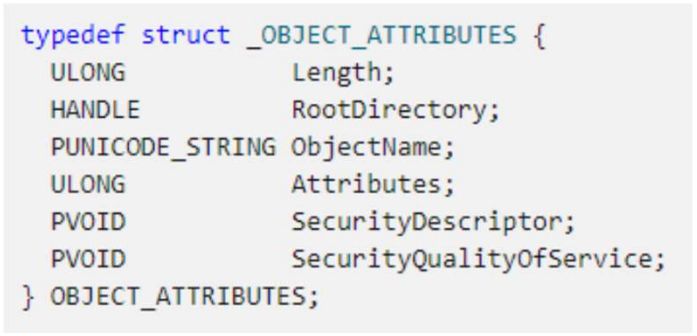
- This allows us to better understand the Windows kernel syscall user-side boundary activities
- This allows us to use native functions that may not be exposed in Kernel32.dll
- This allows our binaries to be as compact as possible (~20-30 times smaller than the smallest C++ binary I could write that does the same function)
- Could be used to bypass *certain* EDR monitoring methods

What do we need to know (4 items)?

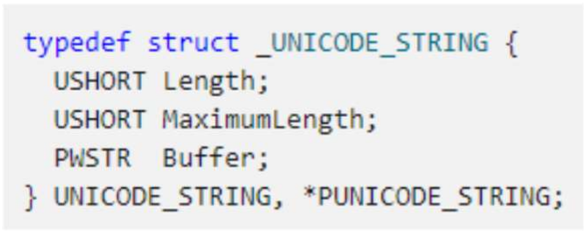
- You need the to know the **SSDT index** to place in EAX prior to the syscall (we can get this from NTDLL.DLL for our version of Windows)
- You need to know **what arguments** the native API function needs (theoretically, this is straightforward inference from the C++ documentation, but there are several low-level gotcha's)
- You need to know the architecture-specific **calling conventions** (ABI) for the architecture (syscall is just an optimized function call to a lower CPL value (Ring 0), which is a higher privilege level)
- You need to know **how to layout the arguments** in the registers and on the stack correctly to make it work. There are several Windows-specific details that are needed including Unicode strings, etc.

(Be sure to add a Defender exclusion as your code may trigger a signature-based quarantine, which becomes annoying after a while)

NtDeleteFile – One of the Simplest Examples



```
typedef struct _OBJECT_ATTRIBUTES {
  ULONG           Length;
  HANDLE          RootDirectory;
  PUNICODE_STRING ObjectName;
  ULONG           Attributes;
  PVOID           SecurityDescriptor;
  PVOID           SecurityQualityOfService;
} OBJECT_ATTRIBUTES;
```



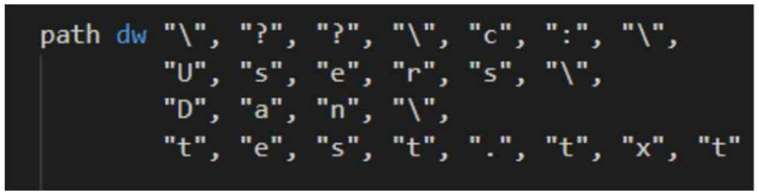
```
typedef struct _UNICODE_STRING {
  USHORT Length;
  USHORT MaximumLength;
  PWSTR  Buffer;
} UNICODE_STRING, *PUNICODE_STRING;
```

- Requires Object Attributes structure
- This structure has 6 elements
- The PUNICODE_STRING has multiple elements
- So, first step is to recreate this structure in assembly…

https://docs.microsoft.com/en-us/windows/win32/api/ntdef/ns-ntdef-_object_attributes

https://docs.microsoft.com/en-us/windows/win32/api/subauth/ns-subauth-unicode_string

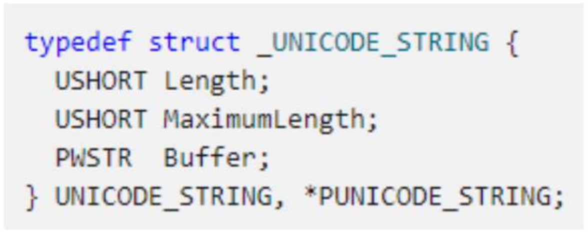
NtDeleteFile – One of the Simplest Examples



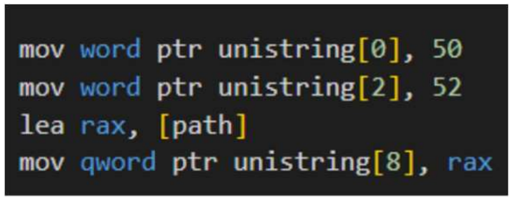
```
path dw "\", "?", "?", "\", "c", ":", "\",
        "U", "s", "e", "r", "s", "\",
        "D", "a", "n", "\",
        "t", "e", "s", "t", ".", "t", "x", "t"
```

- First, let’s address the Unicode path of the file
- Microsoft uses 16-bit Unicode for these strings on my machine, I can declare word-sized Unicode characters this way (using Microsoft Macro Assembler x64)
- The file I want to delete is “c:\Users\Dan\test.txt”

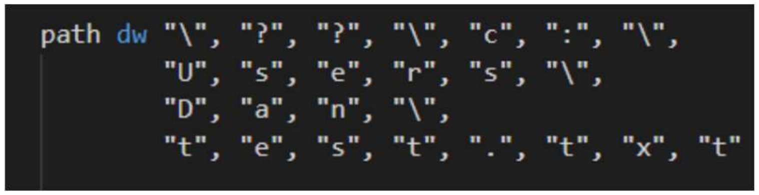
NtDeleteFile – One of the Simplest Examples



```
typedef struct _UNICODE_STRING {
  USHORT Length;
  USHORT MaximumLength;
  PWSTR  Buffer;
} UNICODE_STRING, *PUNICODE_STRING;
```



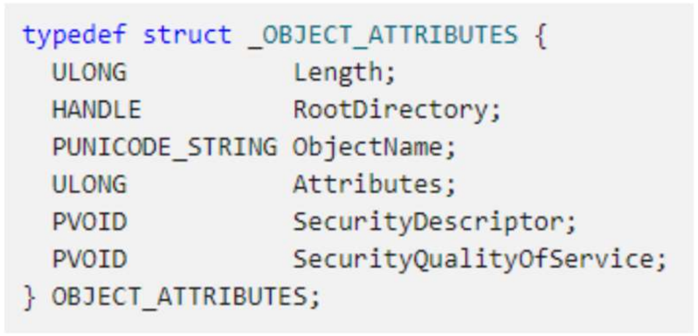
```
mov word ptr unistring[0], 50
mov word ptr unistring[2], 52
lea rax, [path]
mov qword ptr unistring[8], rax
```



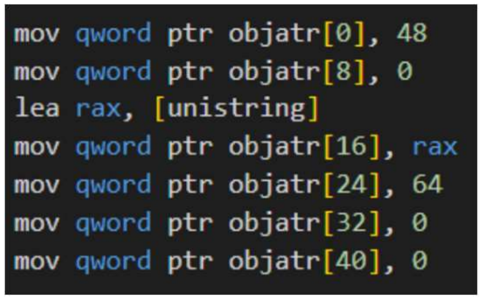
```
path dw "\", "?", "?", "\", "c", ":", "\",
        "U", "s", "e", "r", "s", "\",
        "D", "a", "n", "\",
        "t", "e", "s", "t", ".", "t", "x", "t"
```

- Now that we have the path (Buffer), we need to create a structure with 2 other entries ahead of the pointer to the Buffer (length and max length)
- Here, I am using 50 (decimal) to define the length and 52 (decimal) to define the max length

NtDeleteFile – One of the Simplest Examples



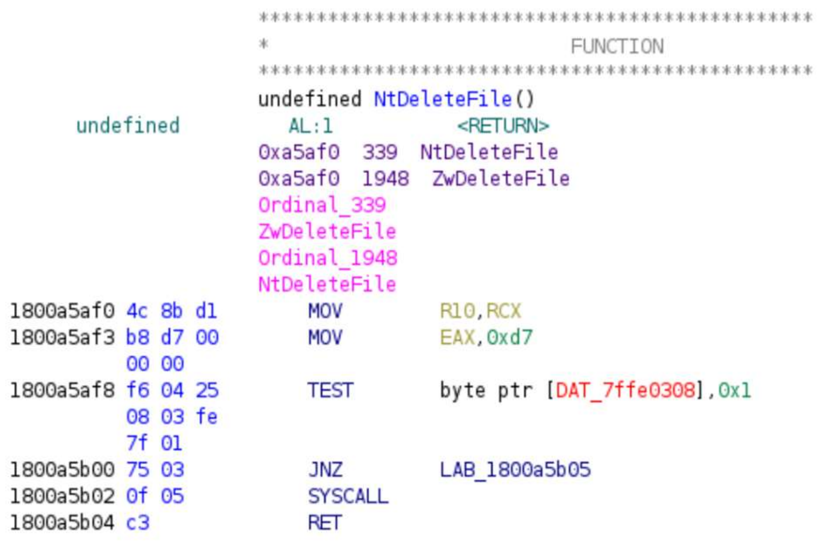
```
typedef struct _OBJECT_ATTRIBUTES {
  ULONG           Length;
  HANDLE          RootDirectory;
  PUNICODE_STRING ObjectName;
  ULONG           Attributes;
  PVOID           SecurityDescriptor;
  PVOID           SecurityQualityOfService;
} OBJECT_ATTRIBUTES;
```



```
mov qword ptr objatr[0], 48
mov qword ptr objatr[8], 0
lea rax, [unistring]
mov qword ptr objatr[16], rax
mov qword ptr objatr[24], 64
mov qword ptr objatr[32], 0
mov qword ptr objatr[40], 0
```

- Now that the UNICODE_STRING structure is created, we can move on to the Object Attributes structure
- Here, I define the Object Attributes as 48 bytes long (Length), null RootDirectory, a pointer to the Unicode String structure I just made (ObjectName), 64 (Attributes), null SecurityDescriptor, and null SecurityQualityOfService

NtDeleteFile – One of the Simplest Examples

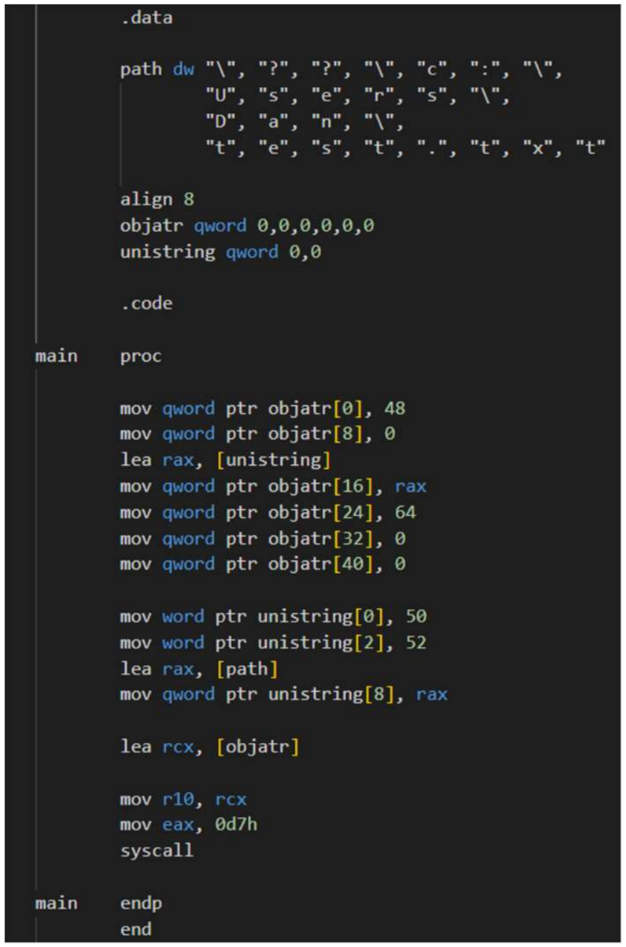


```
                    ****************************************************
                    *                          FUNCTION
                    ****************************************************
                    undefined NtDeleteFile()
      undefined       AL:1            <RETURN>
                    0xa5af0  339  NtDeleteFile
                    0xa5af0  1948  ZwDeleteFile
                    Ordinal_339
                    ZwDeleteFile
                    Ordinal_1948
                    NtDeleteFile
1800a5af0 4c 8b d1       MOV       R10,RCX
1800a5af3 b8 d7 00       MOV       EAX,0xd7
          00 00
1800a5af8 f6 04 25       TEST      byte ptr [DAT_7ffe0308],0x1
          08 03 fe
          7f 01
1800a5b00 75 03          JNZ       LAB_1800a5b05
1800a5b02 0f 05          SYSCALL
1800a5b04 c3             RET
```

- Since we are in a 64-bit environment, we need to pass a pointer to the Object Attributes structure in RCX for the syscall (based on the ABI)

- Now, we need to set up the syscall. We can disassemble NTDLL.DLL for our version of Windows for those specifics for NtDeleteFile

NtDeleteFile – One of the Simplest Examples



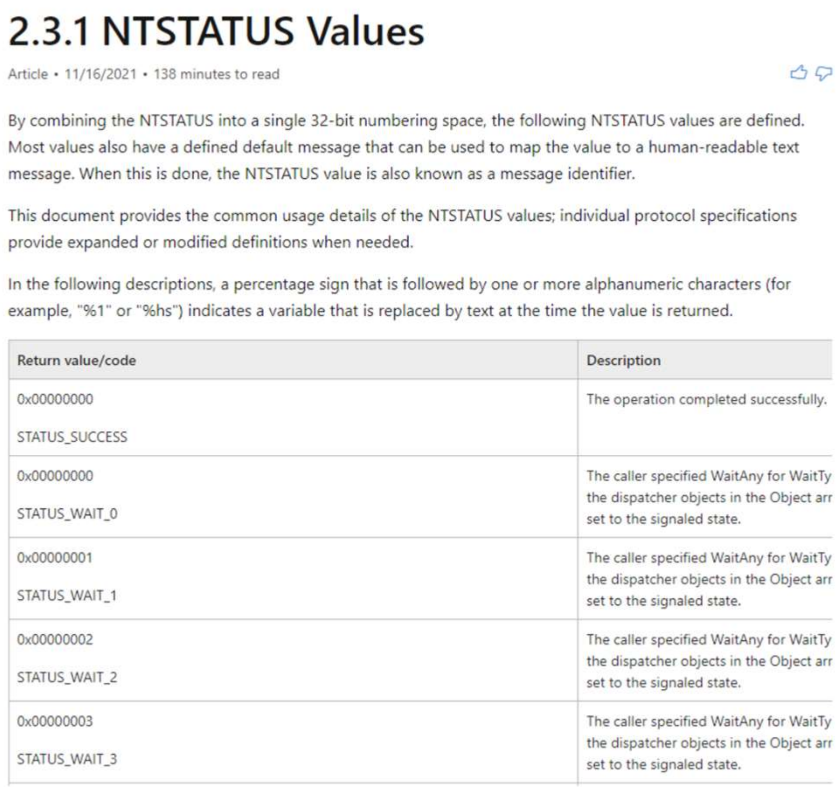
```
        .data

        path dw "\", "?", "?", "\", "c", ":", "\",
                "U", "s", "e", "r", "s", "\",
                "D", "a", "n", "\",
                "t", "e", "s", "t", ".", "t", "x", "t"

        align 8
        objatr qword 0,0,0,0,0,0
        unistring qword 0,0

        .code

main    proc

        mov qword ptr objatr[0], 48
        mov qword ptr objatr[8], 0
        lea rax, [unistring]
        mov qword ptr objatr[16], rax
        mov qword ptr objatr[24], 64
        mov qword ptr objatr[32], 0
        mov qword ptr objatr[40], 0

        mov word ptr unistring[0], 50
        mov word ptr unistring[2], 52
        lea rax, [path]
        mov qword ptr unistring[8], rax

        lea rcx, [objatr]

        mov r10, rcx
        mov eax, 0d7h
        syscall

main    endp
        end
```

- Here is the completed code. (BTW, does not move to Recycle Bin when deleted)

- Don’t forget the 8-byte alignment for x64. Without it, you will receive an 80000002 error in RAX after the syscall (data type misalignment) and it won’t delete the file

- Assemble using: ml64 NTDeleteFile.asm /Zf /link /subsystem:console /entry:main

NTSTATUS Codes



2.3.1 NTSTATUS Values

Article • 11/16/2021 • 138 minutes to read

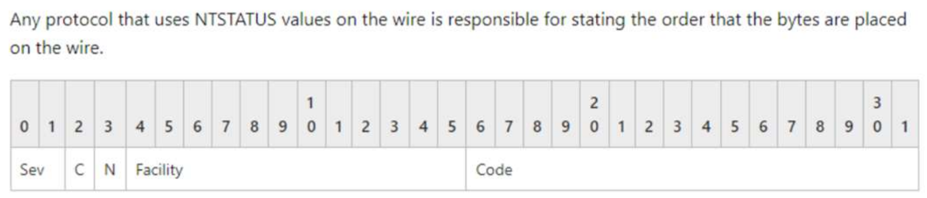
By combining the NTSTATUS into a single 32-bit numbering space, the following NTSTATUS values are defined. Most values also have a defined default message that can be used to map the value to a human-readable text message. When this is done, the NTSTATUS value is also known as a message identifier.

This document provides the common usage details of the NTSTATUS values; individual protocol specifications provide expanded or modified definitions when needed.

In the following descriptions, a percentage sign that is followed by one or more alphanumeric characters (for example, "%1" or "%hs") indicates a variable that is replaced by text at the time the value is returned.

Return value/code	Description
0x00000000 STATUS_SUCCESS	The operation completed successfully.
0x00000000 STATUS_WAIT_0	The caller specified WaitAny for WaitTy the dispatcher objects in the Object arr set to the signaled state.
0x00000001 STATUS_WAIT_1	The caller specified WaitAny for WaitTy the dispatcher objects in the Object arr set to the signaled state.
0x00000002 STATUS_WAIT_2	The caller specified WaitAny for WaitTy the dispatcher objects in the Object arr set to the signaled state.
0x00000003 STATUS_WAIT_3	The caller specified WaitAny for WaitTy the dispatcher objects in the Object arr set to the signaled state.

- Status codes (including errors) after the syscall are returned in RAX. These are NTSTATUS codes.
- Microsoft has a good link that provides a description…
- The code is a compound value:



Any protocol that uses NTSTATUS values on the wire is responsible for stating the order that the bytes are placed on the wire.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
Sev		C	N	Facility												Code															

https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-erref/87fba13e-bf06-450e-83b1-9241dc81e781
https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-erref/596a1078-e883-4972-9bbc-49e60bebca55

NtCreateFile – More Complex – Regs + Stack Args

I have included the code for NtCreateFile. Use what you now know to decipher what is occurring.