

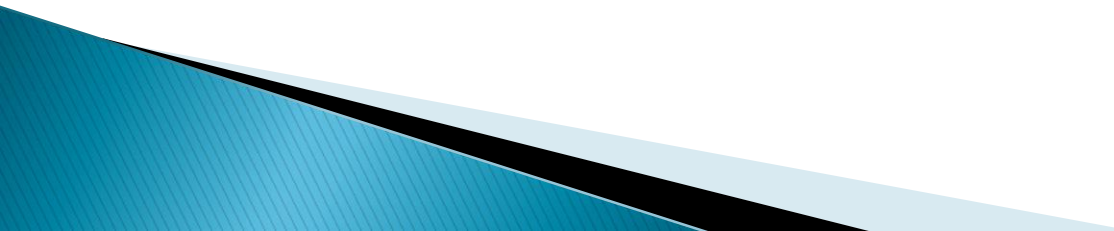
LES AVENTURIERS

DES SYSCALLS ID PERDUS

ESIEA Secure Edition – 20/05/2022

Par Alice Climent-Pommeret

Pourquoi ?

- ▶ Présentation de 3 techniques populaires utilisées pour le contournement d'EDR et d'anti-virus
 - ▶ Utilisateurs : Dev de malware et Red Teamer
- 

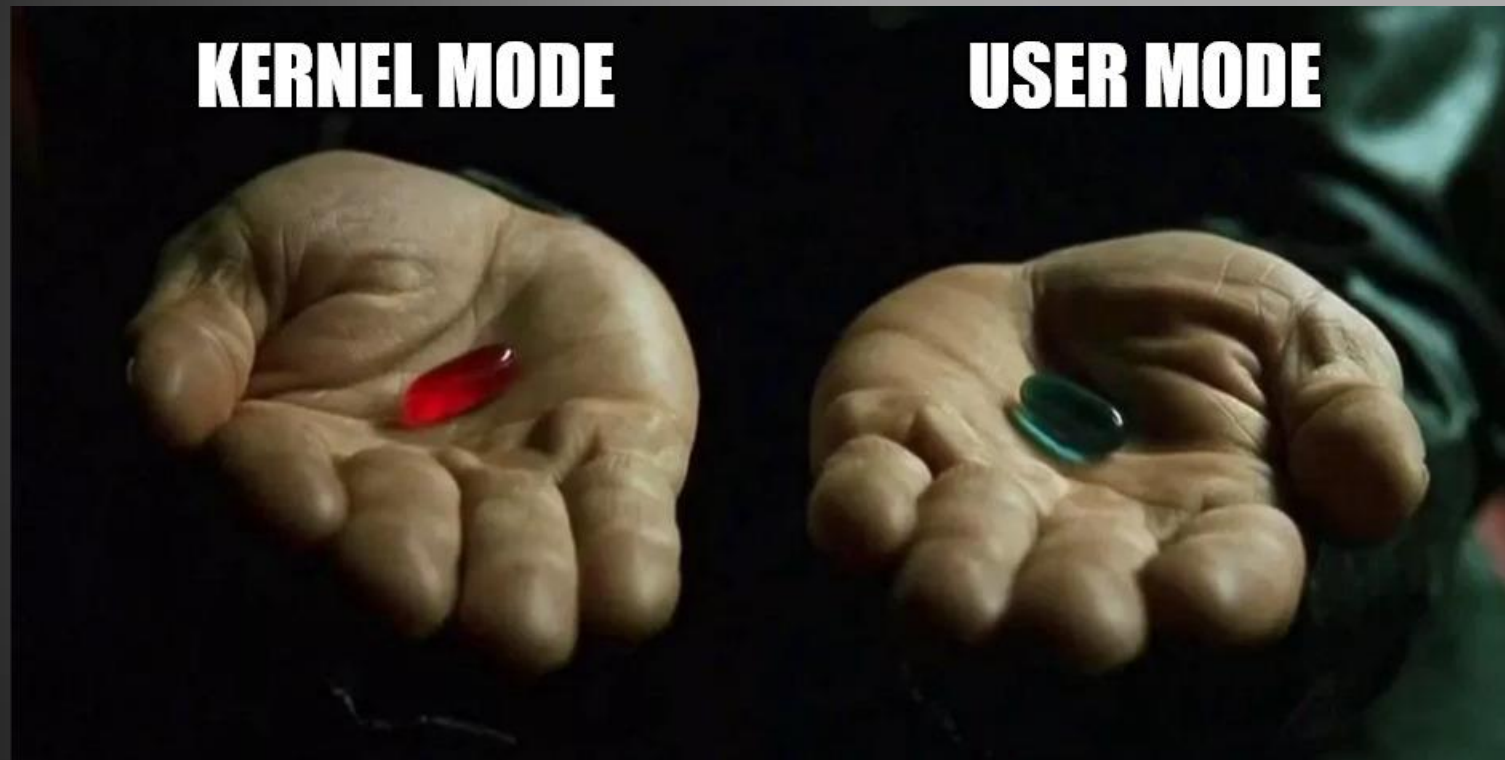
Qui suis-je ?

- ▶ OffSec à la Caisse Nationale de l'Assurance Maladie
- ▶ Malware Analyst à mes heures perdues
- ▶ Experte en baguettes & viennoiseries repentie

Sommaire

- ▶ EDR : User Mode vs Kernel Mode
- ▶ Hook ou la revanche du capitaine crochet
- ▶ Direct Syscall vous avez dit ?
- ▶ Récupérer l'adresse d'une DLL : c'est la base !
- ▶ Dis moi ce que tu exportes, je te dirais où tu es
- ▶ Can you show me some Syscall ID ?

EDR



▶ User mode :

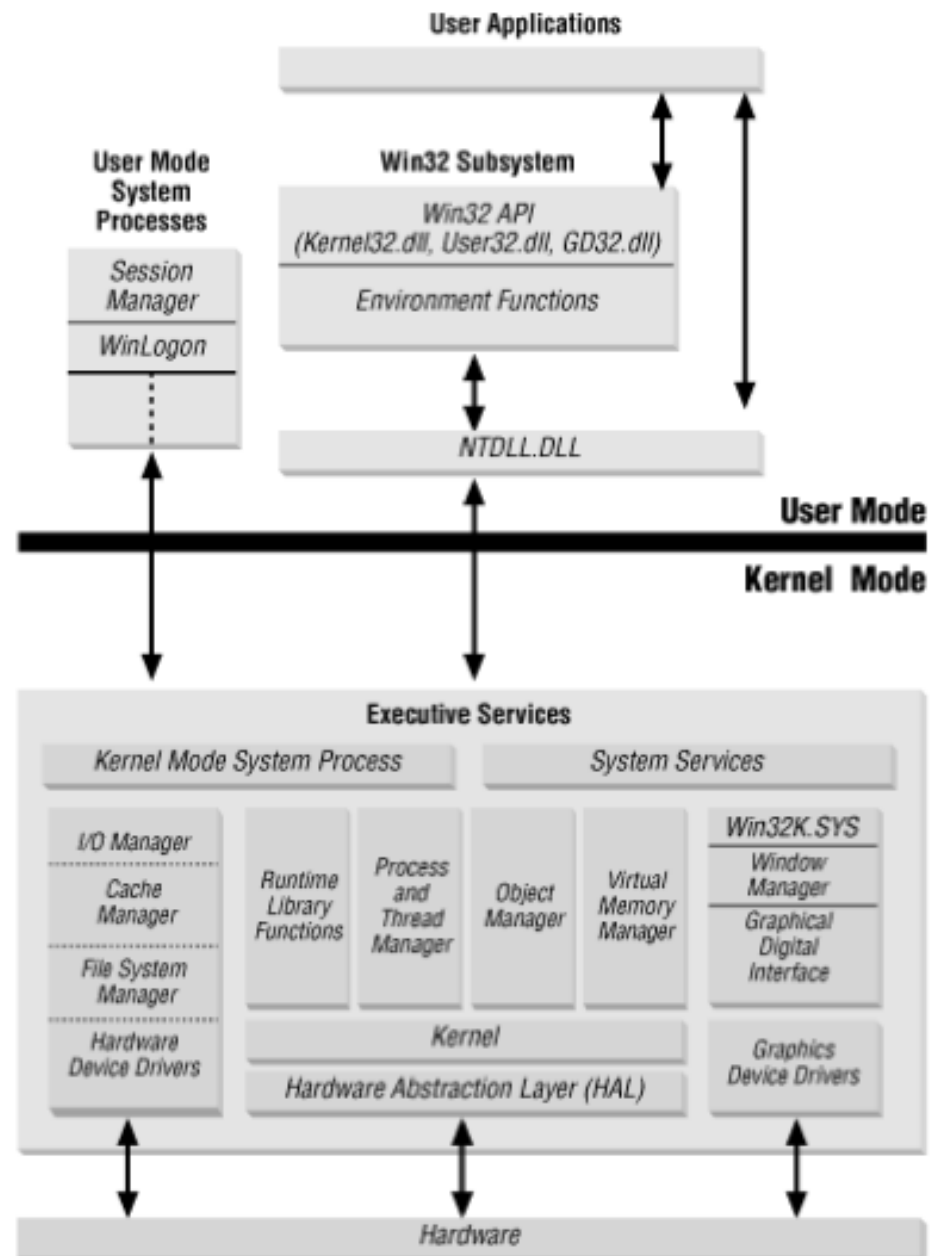
- Environnement avec lequel l'utilisateur interagit (application, API Windows, DLL, etc)
- Espace d'adressage privé

▶ Kernel mode :

- Inaccessible à l'utilisateur (drivers, System Service, processus kernel etc);
- Espace d'adressage unique (pas de segmentation);
- God mode

EDR

- ▶ **Présence au niveau User Mode**
 - Via injection de DLL pour le monitoring de processus
 - Agent de l'EDR
- ▶ **Présence au niveau Kernel Mode**
 - Via l'utilisation de Driver



Hook

ou la revanche du Capitaine Crochet



Qu'est ce qu'un hook ?

- ▶ Permet la redirection du flux d'exécution d'un programme. Dans notre contexte, lors d'un appel à une fonction de l'API Windows
- ▶ Peut être placé dans le code d'une DLL (kernel32, kernelbase, ntdll) ou dans l'Import Address Table (IAT) du processus visé

Dans quel but ?

- ▶ Pour les EDRs :
 - Analyser l'appel d'une fonction afin de déterminer si il est légitime ou malveillant
- ▶ Pour les malwares :
 - Faire exécuter à l'insu de l'utilisateur du code malveillant (vol d'identifiant, keylogging, etc) au sein d'un processus légitime

Exemple



Appel à OpenProcess()

<div>▼ EB 06</div> <div>90 90 90 90 90 90</div>	<div><JMP.&OpenProcess></div> <div>nop nop nop nop nop nop</div>	<div>OpenProcess</div>
<div>▼ FF25 0A150800</div> <div>90 90 90 90 90 90 90 90 90 90 90 90 90 90 90</div>	<div><jmp qword ptr ds:[<&OpenProcess>]</div> <div>nop nop nop nop nop nop nop nop nop nop nop nop nop nop nop</div>	<div>JMP.&OpenProcess</div>

0007723D418 &"Hfîhe3ÉicÀÇ\$00"]=000007FEFD4B3FA0 "Hfîhe3ÉicÀÇ\$00"
 08 kernel32.dll:\$1BF08 #1B708 <JMP.&OpenProcess>

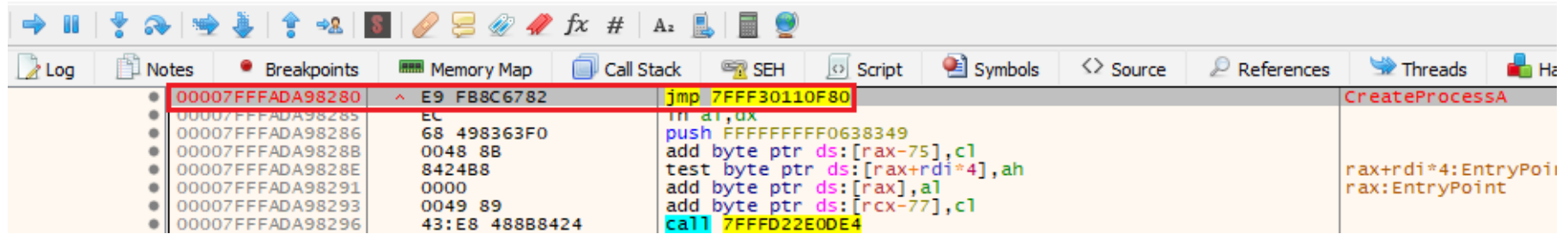
000007FEFD4B3FA0 <kernelbase.OpenProcess>

```

sub     rsp,68
xor     r9d,r9d
movsxd  rax,r8d
mov     dword ptr ss:[rsp+30],30
mov     qword ptr ss:[rsp+28],r9
mov     qword ptr ss:[rsp+20],rax
mov     qword ptr ss:[rsp+38],r9
test    edx,edx
jne     kernelbase.7FEFD4BE840
mov     dword ptr ss:[rsp+48],r9d
mov     qword ptr ss:[rsp+40],r9
mov     qword ptr ss:[rsp+50],r9
mov     qword ptr ss:[rsp+58],r9
mov     ecx,ecx
lea     r9,qword ptr ss:[rsp+20]
lea     r8,qword ptr ss:[rsp+30]
lea     rcx,qword ptr ss:[rsp+88]
call    qword ptr ds:[<&NtOpenProcess>]
test    eax,eax
js      kernelbase.7FEFD4B3A35
        
```


Module: kernelbase.dll

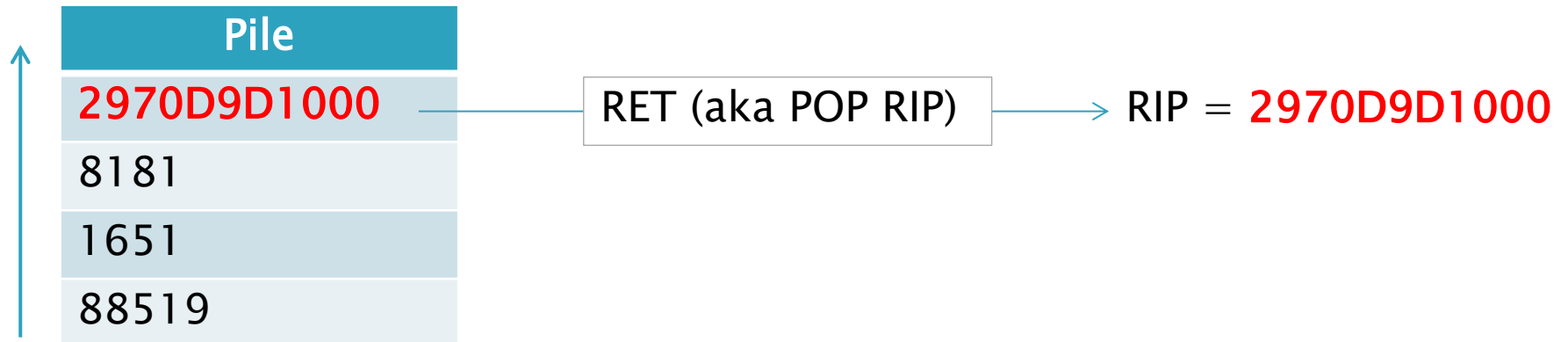
Debug Tracing Plugins Favourites Options Help Nov 24 2021 (TitanEngine)



```

mov rax,2970D9D1000
push rax
mov rax,2970DFB21C0
ret

```



```

48:894C24 40 mov qword ptr ss:[rsp+40],rcx
48:8B4C24 20 mov rcx,qword ptr ss:[rsp+20]
48:8D5424 20 lea rdx,qword ptr ss:[rsp+20]
EC cld
48:B8 C047FA9EFF7F0000 mov rax,atcuf64.7FFF9EFA47C0
FFD0 call rax

```

Base	Module	Party	Path
00007EE65E670000	neh.exe	User	C:\Users\User\Documents\PEB.exe
00007FFF9EFA0000	atcuf64.dll	User	C:\Program Files\Bitdefender Antivirus Free\atcuf\dlls_265575945590319423\atcuf64.dll
00007FFFAABC0000	apphelp.dll	System	C:\Windows\System32\apphelp.dll
00007FFFADA20000	kernelbase.dll	System	C:\Windows\System32\KernelBase.dll
00007FFFAE7C0000	kernel32.dll	System	C:\Windows\System32\kernel32.dll
00007FFB0060000	ntdll.dll	System	C:\Windows\System32\ntdll.dll



« Direct Syscall » vous avez dit ?

- ▶ Kernel32.dll : utilisé pour interagir avec d'autres processus, la mémoire, les disques ou le système de fichier
- ▶ user32.dll et gdi32.dll : utilisé pour interagir avec la GUI Windows.

Les « direct syscalls » sont effectués principalement par :

- ▶ les fonctions Nt* et Zw* au sein de « ntdll.dll »
- ▶ les fonctions NtUser* et NtGdi* au sein de win32u.dll

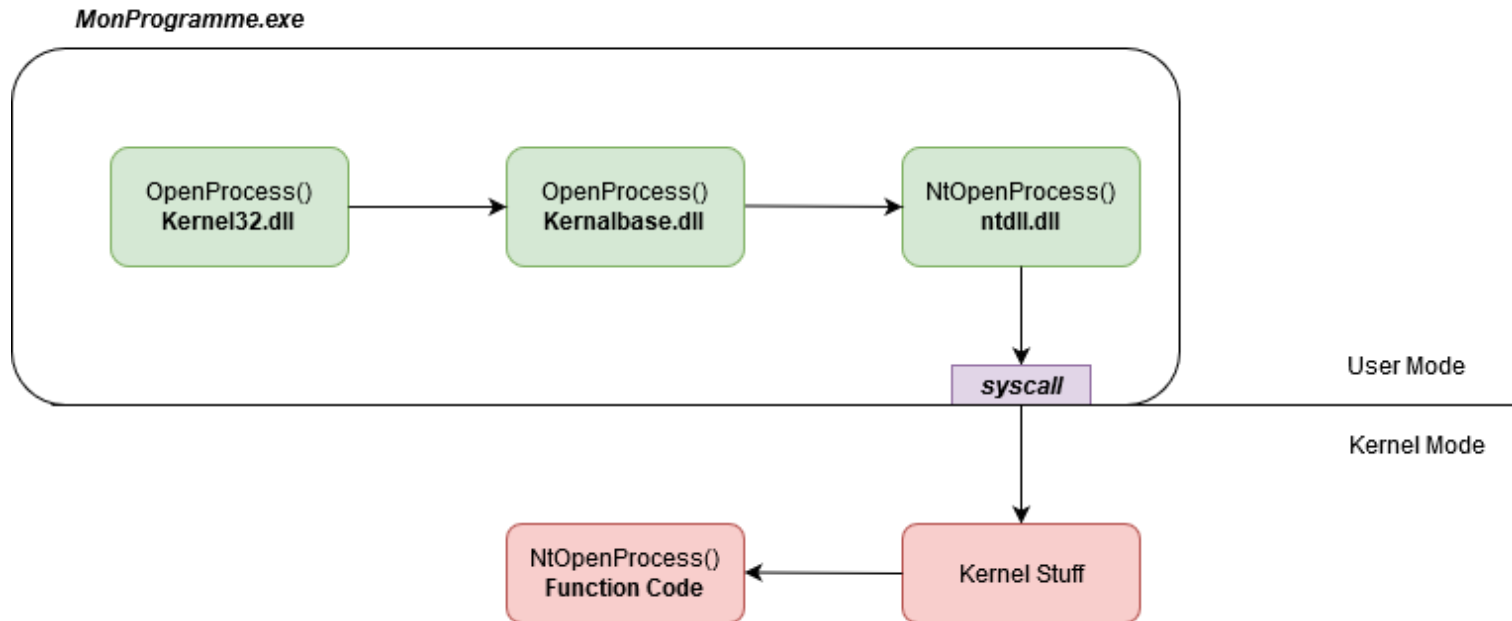
Attention !

- ▶ Toutes les fonctions dans kernel32, user32 ou gdi32 ne mènent pas à un « direct syscall »
- ▶ Le code des fonctions Nt*, Zw*, NtUser* et NtGdi* est situé au niveau du Kernel

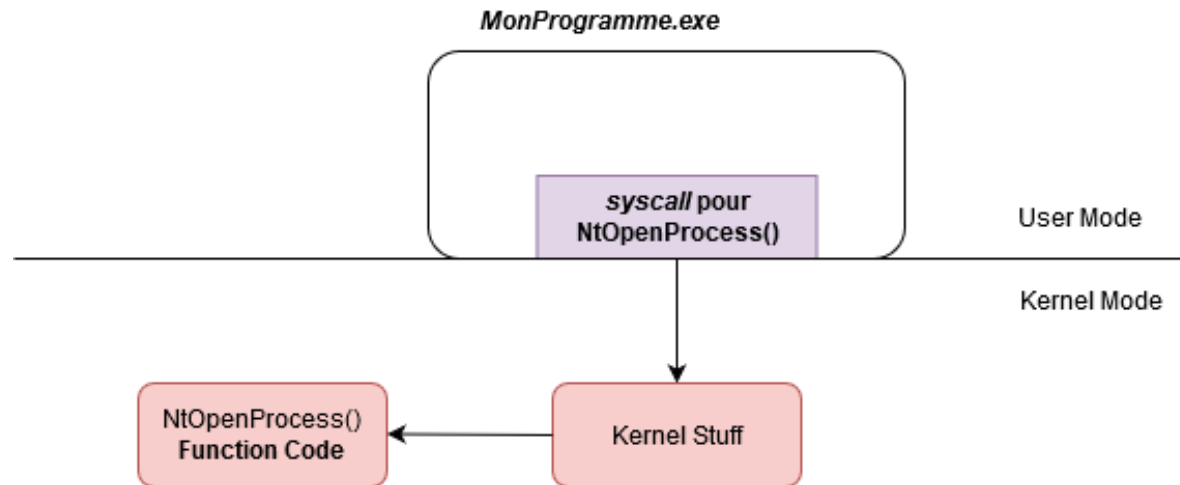
Intérêt du « Direct Syscall »

- ▶ Appel de fonction effectué à la limite du « User Mode »
- ▶ Permet de contourner les différents hooks placés dans les DLL

Appel standard à OpenProcess()



Appel utilisant un Direct Syscall



4C:8BD1	mov r10,rcx	NtOpenProcess
B8 23000000	mov eax,23	23: '#'
OF05	syscall	
C3	ret	

- ▶ Ce code est appelé un « syscall stub »
- ▶ Son rôle est de transférer le flux d'exécution de la fonction au noyau.

4C:8BD1	mov r10,rcx	NtOpenProcess
B8 23000000	mov eax,23 ← Syscall ID	23: '#'
0F05	syscall	
C3	ret	

L'élément le plus important d'un « syscall stub » est son « Syscall ID » connu aussi sous le nom de :

- ▶ System Service Number ou SSN (nom officiel)
- ▶ Syscall Number

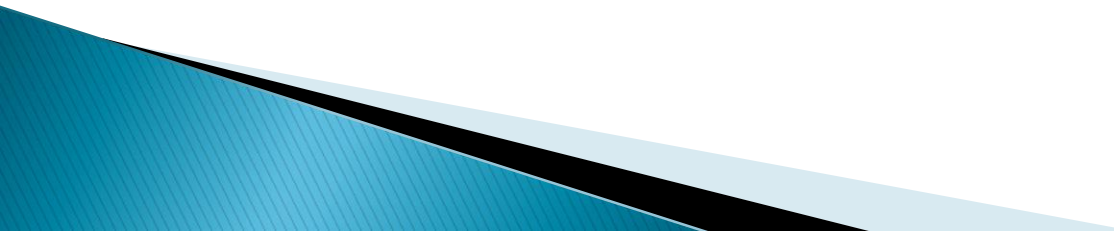


- ▶ L'unique différence entre plusieurs « Syscall Stub » est le « Syscall ID »
- ▶ Le « Syscall ID » est unique pour une version d'OS donnée (change entre version et Service Pack)
- ▶ Il sert d'identifiant permettant de retrouver dans le Kernel le code de notre fonction

4C:8BD1	mov r10,rcx
B8 23000000	mov eax,23
0F05	syscall
C3	ret

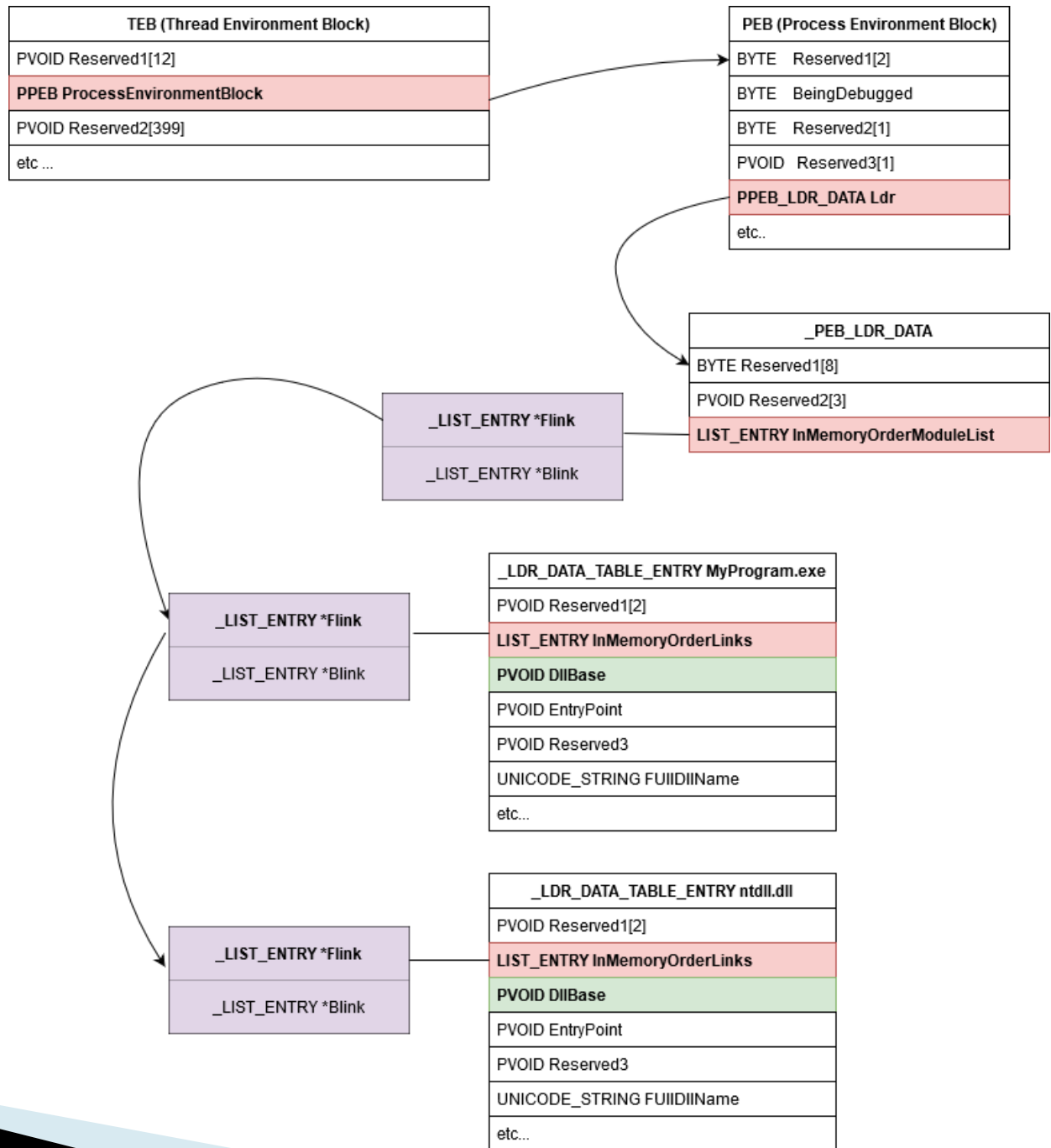
De quoi avons-nous besoin ?

Pour récupérer un « Syscall ID » il nous faut :

- ▶ L'adresse de base de la DLL qui nous intéresse
 - ▶ Dans certains cas, l'adresse de fonctions exportées par cette DLL
- 

Récupérer l'adresse d'une DLL :
c'est la base !

- ▶ Comment récupérer la « base address » d'une DLL :
 - En utilisant la fonction « GetModuleHandle » ou « LoadLibrary »
 - Via la structure de donnée « Process Environment Block »
- ▶ Dans quelle circonstance utiliser la version utilisant la « PEB »
 - Si « GetModuleHandle » est hooké 💀
 - Dans le cadre d'un shellcode



Dis moi ce que tu exportes,
Je te dirais où tu es

- ▶ Comment récupérer l'adresse d'une fonction exportée par une DLL :
 - En utilisant la fonction « GetProcAddress »
 - Via l' « Export Address Table » de la DLL
- ▶ Dans quelle circonstance utiliser la version utilisant l'« EAT »
 - Si « GetProcAddress » est hooké 💀
 - Dans le cadre d'un shellcode

▶ Address Of Function

- Tableau contenant les RVA (Relative Virtual Address) des fonctions exportées (Export Address Table)

▶ Address Of Names

- Tableau contenant les noms de fonctions exportées (Export Name Table)

▶ Address Of Name Ordinals

- Tableau contenant les ordinaux

_Image_Export_Directory
DWORD Characteristics
DWORD TimeDateStamp
WORD MajorVersion
WORD MinorVersion
DWORD Name
DWORD Base
DWORD NumberOfFunctions
DWORD NumberOfNames
DWORD AddressOfFunctions
DWORD AddressOfNames
DWORD AddressOfNameOrdinals

AddressOfFunctions	7F110	40230	41060	410A0
--------------------	-------	-------	-------	-------

AddressOfNameOrdinals	01	02	03	04
-----------------------	----	----	----	----

AddressOfNames	A_SHAFinal	A_SHAInit	A_SHA_Update	AlpcAdjustCompletionListConcurrencyCount
----------------	------------	-----------	--------------	--



- ▶ Pour retrouver l'adresse réelle de la fonction dans le contexte du processus

Base Address (via la PEB) + RVA (via l'EAT)

- ▶ Habituellement :
 - `GetProcAddress(GetModuleHandle(« ntdll.dll »), « A_SHAInit »)`
 - `GetProcAddress(LoadLibrary(« NotLoaded.dll »), « HelloWorld »)`

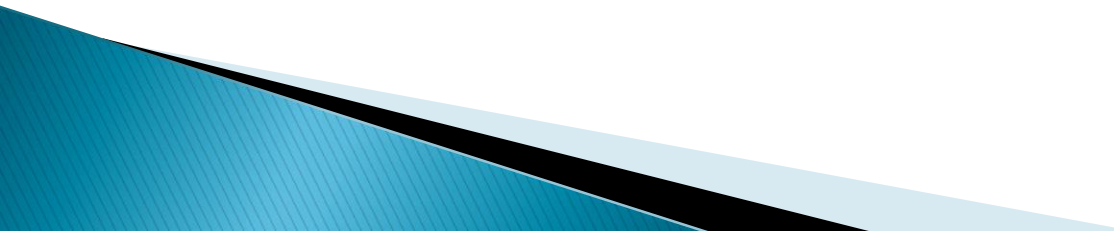
Can you show me some Syscall ID ?



- ▶ Hell's Gate : @am0nsec et @RtlMateusz
 - <https://github.com/am0nsec/HellsGate>
- ▶ Halo's Gate : Reenz0h of Sektor7
 - <https://institute.sektor7.net/rto-win-evasion>
- ▶ SysWhispers2 : @Jackson_T & @modexpblog
 - <https://www.mdsec.co.uk/2020/12/bypassing-user-mode-hooks-and-direct-invocation-of-system-calls-for-red-teams/>
 - <https://github.com/jthuraisamy/SysWhispers2>

Hell's Gate



- ▶ Permet d'intégrer au sein des programmes des « direct syscall » pour des fonctions données.
 - ▶ Récupère les « syscall ID » en recherchant le pattern d'un « syscall stub » au sein de « ntdll »
 - ▶ Dans le PoC se limite à « NtAllocateVirtualMemory », « NtProtectVirtualMemory », « NtCreateThreadEx » et « NtWaitForSingleObject »
 - ▶ Ne fonctionne pas en cas de hook sur la fonction ciblée
- 

```

if (*((PBYTE)pFunctionAddress + cw) == 0x4c
    && *((PBYTE)pFunctionAddress + 1 + cw) == 0x8b
    && *((PBYTE)pFunctionAddress + 2 + cw) == 0xd1
    && *((PBYTE)pFunctionAddress + 3 + cw) == 0xb8
    && *((PBYTE)pFunctionAddress + 6 + cw) == 0x00
    && *((PBYTE)pFunctionAddress + 7 + cw) == 0x00) {
    BYTE high = *((PBYTE)pFunctionAddress + 5 + cw);
    BYTE low = *((PBYTE)pFunctionAddress + 4 + cw);
    pVxTableEntry->wSystemCall = (high << 8) | low;
    break;
}

```

```

mov r10,rcx          // 0x4c 0x8b 0xd1
mov eax, SyscallNumber // 0xb8 0xZZ 0xZZ 0x00 0x00

```


Halo's Gate

- ▶ Evolution de Hell's Gate
 - Permet d'intégrer au sein des programmes des « direct syscall » pour des fonctions données.
 - Dans le PoC se limite à « NtAllocateVirtualMemory », « NtProtectVirtualMemory », « NtCreateThreadEx » et « NtWaitForSingleObject »
 - Récupère les « syscall ID » en recherchant le pattern d'un « syscall stub » au sein de « ntdll »

- ▶ Fonctionne même en cas de hook

- ▶ Pour le contournement de hook, utilise une recherche de proche en proche pour identifier le « syscall ID »

Syscall ID incrémentaux

00000000779D1460	4C:8BD1	mov r10,rcx	ZwQueryDefaultLocale
00000000779D1463	B8 12000000	mov eax,12	
00000000779D1468	0F05	syscall	
00000000779D146A	C3	ret	
00000000779D146B	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
00000000779D1470	4C:8BD1	mov r10,rcx	NtQueryKey
00000000779D1473	B8 13000000	mov eax,13	
00000000779D1478	0F05	syscall	
00000000779D147A	C3	ret	
00000000779D147B	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
00000000779D1480	4C:8BD1	mov r10,rcx	NtQueryValueKey
00000000779D1483	B8 14000000	mov eax,14	
00000000779D1488	0F05	syscall	
00000000779D148A	C3	ret	
00000000779D148B	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
00000000779D1488	4C:8BD1	mov r10,rcx	ZwAllocateVirtualMemory
00000000779D1493	B8 15000000	mov eax,15	
00000000779D1498	0F05	syscall	
00000000779D149A	C3	ret	
00000000779D149B	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
00000000779D14A0	4C:8BD1	mov r10,rcx	NtQueryInformationProcess
00000000779D14A3	B8 16000000	mov eax,16	
00000000779D14A8	0F05	syscall	
00000000779D14AA	C3	ret	
00000000779D14AB	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
00000000779D1480	4C:8BD1	mov r10,rcx	ZwWaitForMultipleObjects32
00000000779D1483	B8 17000000	mov eax,17	
00000000779D1488	0F05	syscall	
00000000779D148A	C3	ret	
00000000779D148B	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
00000000779D14C0	4C:8BD1	mov r10,rcx	ZwWriteFileGather
00000000779D14C3	B8 18000000	mov eax,18	
00000000779D14C8	0F05	syscall	
00000000779D14CA	C3	ret	
00000000779D14CB	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
00000000779D14D0	4C:8BD1	mov r10,rcx	NtSetInformationProcess
00000000779D14D3	B8 19000000	mov eax,19	
00000000779D14D8	0F05	syscall	
00000000779D14DA	C3	ret	

Départ similaire

```
if (*((PBYTE)pFunctionAddress + cw) == 0x4c
    && *((PBYTE)pFunctionAddress + 1 + cw) == 0x8b
    && *((PBYTE)pFunctionAddress + 2 + cw) == 0xd1
    && *((PBYTE)pFunctionAddress + 3 + cw) == 0xb8
    && *((PBYTE)pFunctionAddress + 6 + cw) == 0x00
    && *((PBYTE)pFunctionAddress + 7 + cw) == 0x00) {
    BYTE high = *((PBYTE)pFunctionAddress + 5 + cw);
    BYTE low = *((PBYTE)pFunctionAddress + 4 + cw);
    pVxTableEntry->wSystemCall = (high << 8) | low;
    break;
}
```

Si hooked, on recherche !

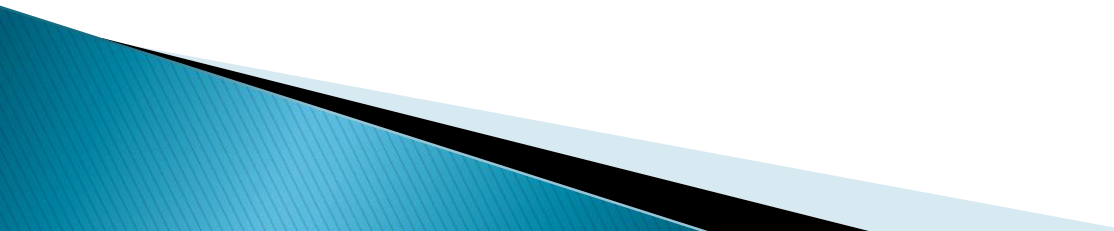
```
int GoUp -32;
int GoDown 32;
// If the first instruction of the syscall is a an unconditional jump (aka it's hooked)
if (*((PBYTE)pFunctionAddress) == 0xe9) {
    // Search beginning pattern of syscall stub through 500 function up and down from our
    for (WORD index = 1; index <= 500; index++) {
        // Search the begining of a syscall stub in the next function down
        if (*((PBYTE)pFunctionAddress + index * GoDown) == 0x4c
            && *((PBYTE)pFunctionAddress + 1 + index * GoDown) == 0x8b
            && *((PBYTE)pFunctionAddress + 2 + index * GoDown) == 0xd1
            && *((PBYTE)pFunctionAddress + 3 + index * GoDown) == 0xb8
            && *((PBYTE)pFunctionAddress + 6 + index * GoDown) == 0x00
            && *((PBYTE)pFunctionAddress + 7 + index * GoDown) == 0x00) {
            BYTE high = *((PBYTE)pFunctionAddress + 5 + index * GoDown);
            BYTE low = *((PBYTE)pFunctionAddress + 4 + index * GoDown);
            // subtract the index from the current syscall identifier to find the one of
            pVxTableEntry->wSystemCall = (high << 8) | low - index;
            return TRUE;
        }
    }
}
```

```
mov r10,rcx                // 0x4c 0x8b 0xd1
mov eax, SyscallNumber     // 0xb8 0xZZ 0xZZ 0x00 0x00
```

Syscall ID incrémentaux

00000000779D1460	4C: 8BD1	mov r10,rcx	ZwQueryDefaultLocale
00000000779D1463	B8 12000000	mov eax,12	
00000000779D1468	0F05	syscall	
00000000779D146A	C3	ret	
00000000779D146B	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
00000000779D1470	4C: 8BD1	mov r10,rcx	NtQueryKey
00000000779D1473	B8 13000000	mov eax,13	
00000000779D1478	0F05	syscall	
00000000779D147A	C3	ret	
00000000779D147B	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
00000000779D1480	4C: 8BD1	mov r10,rcx	NtQueryValueKey
00000000779D1483	B8 14000000	mov eax,14	
00000000779D1488	0F05	syscall	
00000000779D148A	C3	ret	
00000000779D148B	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
00000000779D1490	4C: 8BD1	HOOK	ZwAllocateVirtualMemory
00000000779D1493	B8 15000000	HOOK	
00000000779D1498	0F05	syscall	
00000000779D149A	C3	ret	
00000000779D149B	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
00000000779D14A0	4C: 8BD1	HOOK	NtQueryInformationProcess
00000000779D14A3	B8 16000000	HOOK	
00000000779D14A8	0F05	syscall	
00000000779D14AA	C3	ret	
00000000779D14AB	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
00000000779D14B0	4C: 8BD1	HOOK	ZwWaitForMultipleObjects32
00000000779D14B3	B8 17000000	HOOK	
00000000779D14B8	0F05	syscall	
00000000779D14BA	C3	ret	
00000000779D14BB	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
00000000779D14C0	4C: 8BD1	mov r10,rcx	ZwWriteFileGather
00000000779D14C3	B8 18000000	mov eax,18	
00000000779D14C8	0F05	syscall	
00000000779D14CA	C3	ret	
00000000779D14CB	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
00000000779D14D0	4C: 8BD1	mov r10,rcx	NtSetInformationProcess
00000000779D14D3	B8 19000000	mov eax,19	
00000000779D14D8	0F05	syscall	
00000000779D14DA	C3	ret	

Syswhispers2

- ▶ Ne parse pas le code de « ntdll » mais l'« EAT » (pas d'interaction avec le code)
 - ▶ Fonctionne même avec la présence de hook
 - ▶ Cible les fonctions « Zw* »
 - ▶ Utilise de l'API hashing (EDR, AV evasion)
- 

Différence entre fonctions « Zw » et « Nt »

- ▶ Le code des fonctions « Nt » et « Zw » est au niveau Kernel
- ▶ Depuis le User Mode
 - « Nt » et « Zw » se comporte exactement de la même façon
- ▶ Depuis le Kernel Mode
 - L'appel à une fonction « Zw » par un driver indique que les paramètres viennent d'une source sûre et ne nécessite pas de validation
 - L'appel à une fonction « Nt » par un driver indique que les paramètres peuvent venir d'une source située en User Mode et nécessite une validation

AddressOfFunctions	7F110	40230	41060	410A0
--------------------	-------	-------	-------	-------

AddressOfNameOrdinals	01	02	03	04
-----------------------	----	----	----	----

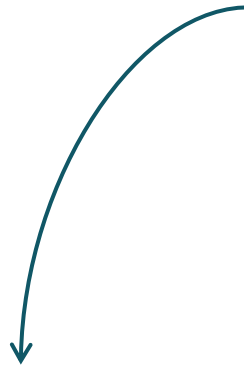
AddressOfNames	A_SHAFinal	A_SHAInit	A_SHA_Update	AlpcAdjustCompletionListConcurrencyCount
----------------	------------	-----------	--------------	--



Adresse	Type	Ordinal	Symbole
0000000077411340	Export	341	NtMapUserPhysicalPagesScatter
0000000077411350	Export	565	NtWaitForSingleObject
0000000077411360	Export	217	NtCallbackReturn
0000000077411370	Export	447	NtReadFile
0000000077411380	Export	280	NtDeviceIoControlFile
0000000077411390	Export	570	NtWriteFile
00000000774113A0	Export	461	NtRemoveIoCompletion
00000000774113B0	Export	459	NtReleaseSemaphore
00000000774113C0	Export	469	NtReplyWaitReceivePort
00000000774113D0	Export	468	NtReplyPort
00000000774113E0	Export	509	NtSetInformationThread
00000000774113F0	Export	497	NtSetEvent
0000000077411400	Export	223	NtClose
0000000077411410	Export	422	NtQueryObject
0000000077411420	Export	405	NtQueryInformationFile
0000000077411430	Export	356	NtOpenKey
0000000077411440	Export	292	NtEnumerateValueKey
0000000077411450	Export	295	NtFindAtom
0000000077411460	Export	395	NtQueryDefaultLocale
0000000077411470	Export	418	NtQueryKey
0000000077411480	Export	440	NtQueryValueKey
0000000077411490	Export	192	NtAllocateVirtualMemory
00000000774114A0	Export	408	NtQueryInformationProcess
00000000774114B0	Export	564	NtWaitForMultipleObjects32
00000000774114C0	Export	571	NtWriteFileGather
00000000774114D0	Export	507	NtSetInformationProcess
00000000774114E0	Export	243	NtCreateKey
00000000774114F0	Export	304	NtFreeVirtualMemory
0000000077411500	Export	320	NtImpersonateClientOfPort
0000000077411510	Export	458	NtReleaseMutant
0000000077411520	Export	411	NtQueryInformationToken
0000000077411530	Export	473	NtRequestWaitReplyPort
0000000077411540	Export	441	NtQueryVirtualMemory
0000000077411550	Export	373	NtOpenThreadToken
0000000077411560	Export	410	NtQueryInformationThread
0000000077411570	Export	364	NtOpenProcess

0000000077411340	4C:8BD1	mov r10,rcx	ZwMapUserPhysicalPagesScatter
0000000077411343	B8 00000000	mov eax,0	
0000000077411348	0F05	syscall	
000000007741134A	C3	ret	
000000007741134B	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
0000000077411350	4C:8BD1	mov r10,rcx	ZwWaitForSingleObject
0000000077411353	B8 01000000	mov eax,1	
0000000077411358	0F05	syscall	
000000007741135A	C3	ret	
000000007741135B	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
0000000077411360	4C:8BD1	mov r10,rcx	NtCallbackReturn
0000000077411363	B8 02000000	mov eax,2	
0000000077411368	0F05	syscall	
000000007741136A	C3	ret	
000000007741136B	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
0000000077411370	4C:8BD1	mov r10,rcx	ZwReadFile
0000000077411373	B8 03000000	mov eax,3	
0000000077411378	0F05	syscall	
000000007741137A	C3	ret	
000000007741137B	0F1F4400 00	nop dword ptr ds:[rax+rax],eax	
0000000077411380	4C:8BD1	mov r10,rcx	ZwDeviceIoControlFile
0000000077411383	B8 04000000	mov eax,4	
0000000077411388	0F05	syscall	
000000007741138A	C3	ret	

ntdll.dll		
Export Name Table	Export sequence number table	Export Address Table
NtCreateFile	8	00138
NtDeleteFile	12	00123
NtFindAtom	26	00208
NtLoadKey	29	00166
NtOpenFile	47	00193
NtQueryKey	61	00184



index	Stub_address	Function_name
0	00123	b5aaf3
1	00138	28c86
2	00166	cac18
3	00184	f1159
4	00193	2b173
5	00208	57d4d

```

EXTERN_C DWORD SW2_GetSyscallNumber(DWORD FunctionHash)
{
    // Ensure SW2_SyscallList is populated.
    if (!SW2_PopulateSyscallList()) return -1;

    for (DWORD i = 0; i < SW2_SyscallList.Count; i++)
    {
        if (FunctionHash == SW2_SyscallList.Entries[i].Hash)
        {
            return i;
        }
    }
}

```

En résumé

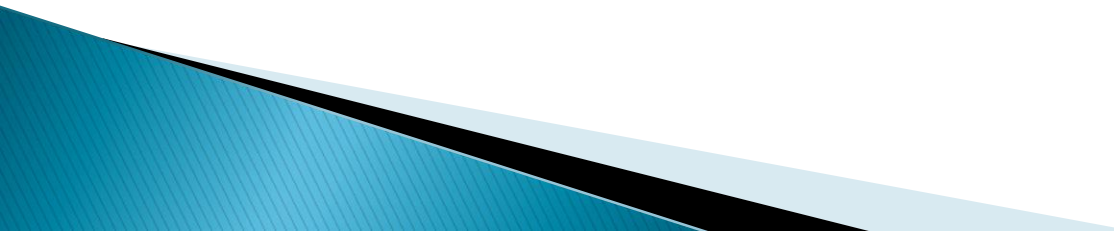
▶ Hell's Gate

- Parse le « syscall stub » pour trouver un pattern de début et récupère le « syscall ID »
- Ne fonctionne pas si hook

▶ Halo's Gate

- Evolution de Hell's gate.
- Si hook, déduit le « syscall ID » en fonction de ses proches

▶ Syswhispers2

- Récupère les « syscalls ID » via l'EAT de la « DLL »
 - Similaire à FreshyCalls
 - Utilise de l'API Hashing
 - Cible les fonctions « Zw »
- 

Questions ?

Pour en savoir plus:

<https://alice.climent-pommeret.red/posts/direct-syscalls-hells-halos-syswhispers2/>