**Topic** :- How Malwares evade  API logging tools.

**Overview** :- This article will discuss a simple trick Malware use to evade API logging. This is a small part of the process injection process performed by the Lokibot malware.

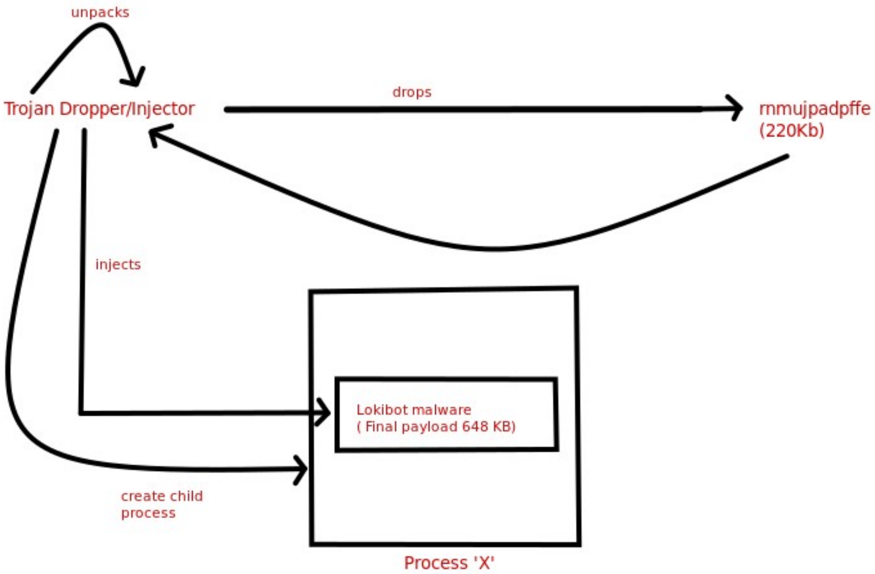
## Malware source :-

- **Sha1 :-** 70c34a5e1442816c23d78454edc2c7505f43f82b
- **Sha256 :-** 563818872af4977ebccd2bc8f97e968edeb6cce444c7a380b3c69e53fd317c2e

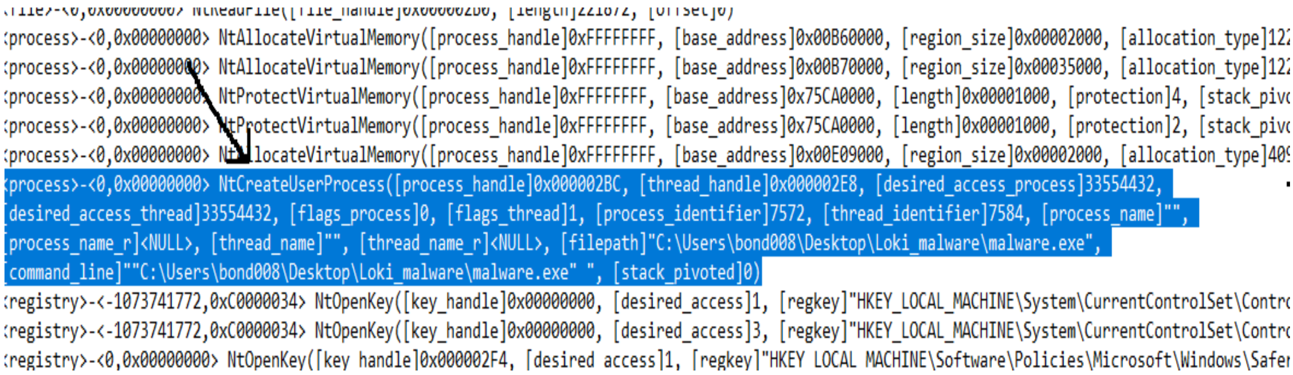
**Tools Used :-** Windbg, Cutter(rizin engine).

API logging can be and is used by antivirus or other detection tools for detecting malicious behaviour. Sometimes user mode hooks are placed to detect which API is being called. Tools like APIMiner uses usermode hook to log Windows API. While analyzing Loki malware, I came upon a neat trick used by the malware to evade API logging by tools. So lets see how it does it.

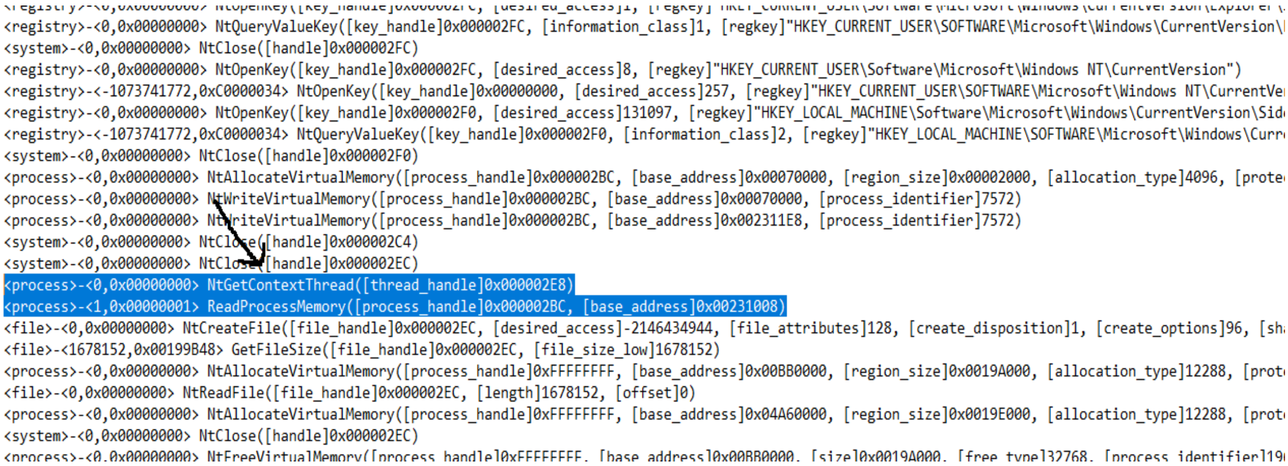
Loki malware is an info stealer malware and uses process injection to infect other process. In this case it creates a new process and injects the Final Payload into it. But it does so in a tricky way by using “sysenter” instructions instead of directly calling the Windows API’s. Below we can see the process injection part of  life cycle of the malware.



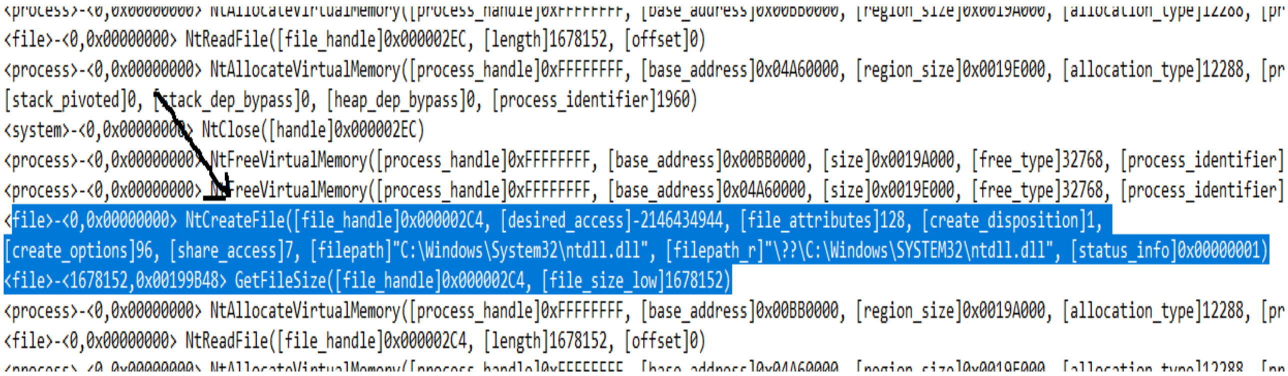
To get  a better idea of the API’s beinng used, we can use APIMiner tool in this case and see what API’s are being called by the sample which will give us some idea of what the malware sample is doing.



```
<file>-<0,0x00000000> NtReadFile([file_handle]0x000002B0, [length]221872, [offset]0)
<process>-<0,0x00000000> NtAllocateVirtualMemory([process_handle]0xFFFFFFFF, [base_address]0x00B60000, [region_size]0x00002000, [allocation_type]12
<process>-<0,0x00000000> NtAllocateVirtualMemory([process_handle]0xFFFFFFFF, [base_address]0x00B70000, [region_size]0x00035000, [allocation_type]12
<process>-<0,0x00000000> NtProtectVirtualMemory([process_handle]0xFFFFFFFF, [base_address]0x75CA0000, [length]0x00001000, [protection]4, [stack_piv
<process>-<0,0x00000000> NtProtectVirtualMemory([process_handle]0xFFFFFFFF, [base_address]0x75CA0000, [length]0x00001000, [protection]2, [stack_piv
<process>-<0,0x00000000> NtAllocateVirtualMemory([process_handle]0xFFFFFFFF, [base_address]0x00E09000, [region_size]0x00002000, [allocation_type]40
<process>-<0,0x00000000> NtCreateUserProcess([process_handle]0x000002BC, [thread_handle]0x000002E8, [desired_access_process]33554432,
[desired_access_thread]33554432, [flags_process]0, [flags_thread]1, [process_identifier]7572, [thread_identifier]7584, [process_name]"",
[process_name_r]<NULL>, [thread_name]"", [thread_name_r]<NULL>, [filepath]"C:\Users\bond008\Desktop\Loki_malware\malware.exe",
[command_line]""C:\Users\bond008\Desktop\Loki_malware\malware.exe" ", [stack_pivoted]0)
<registry>-<-1073741772,0xC0000034> NtOpenKey([key_handle]0x00000000, [desired_access]1, [regkey]"HKEY_LOCAL_MACHINE\System\CurrentControlSet\Contr
<registry>-<-1073741772,0xC0000034> NtOpenKey([key_handle]0x00000000, [desired_access]3, [regkey]"HKEY_LOCAL_MACHINE\System\CurrentControlSet\Contr
<registry>-<0,0x00000000> NtOpenKey([key_handle]0x000002F4, [desired_access]1, [regkey]"HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows\Safer
```



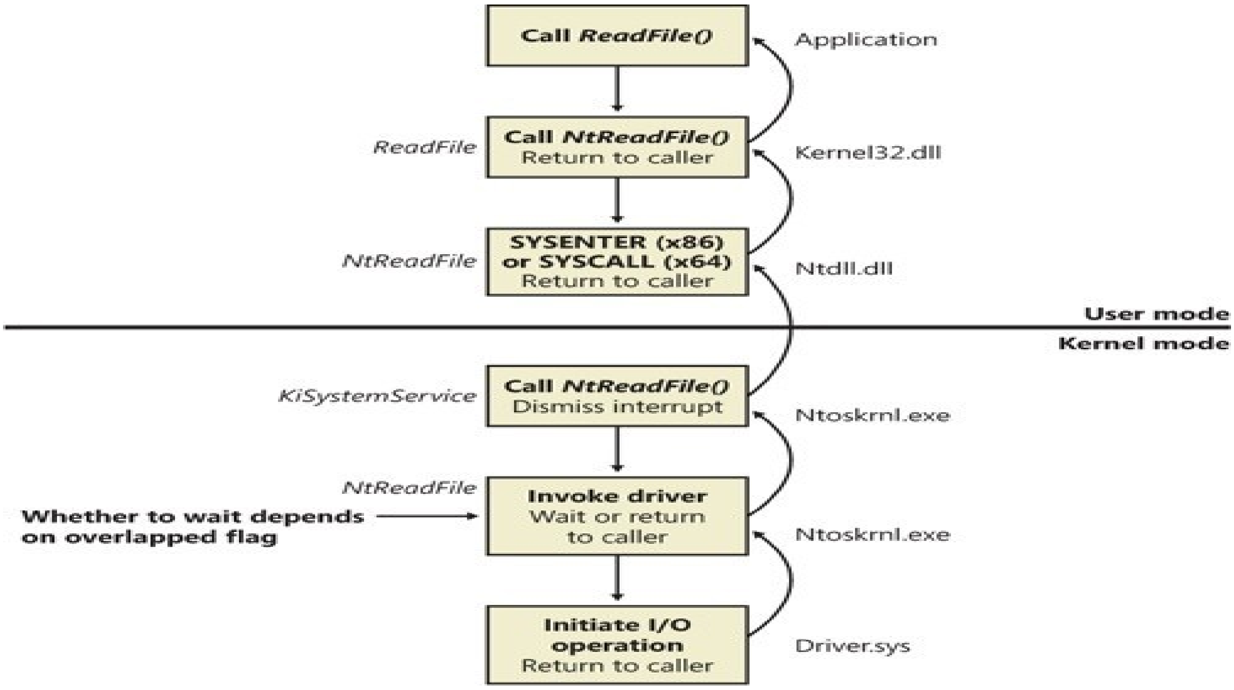
```
<registry>-<0,0x00000000> NtOpenKey([key_handle]0x000002FC, [desired_access]1, [regkey]"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\
<registry>-<0,0x00000000> NtQueryValueKey([key_handle]0x000002FC, [information_class]1, [regkey]"HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\
<system>-<0,0x00000000> NtClose([handle]0x000002FC)
<registry>-<0,0x00000000> NtOpenKey([key_handle]0x000002FC, [desired_access]8, [regkey]"HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion")
<registry>-<-1073741772,0xC0000034> NtOpenKey([key_handle]0x00000000, [desired_access]257, [regkey]"HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows NT\CurrentVe
<registry>-<0,0x00000000> NtOpenKey([key_handle]0x000002F0, [desired_access]131097, [regkey]"HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Sid
<registry>-<-1073741772,0xC0000034> NtQueryValueKey([key_handle]0x000002F0, [information_class]2, [regkey]"HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Curr
<system>-<0,0x00000000> NtClose([handle]0x000002F0)
<process>-<0,0x00000000> NtAllocateVirtualMemory([process_handle]0x000002BC, [base_address]0x00070000, [region_size]0x00002000, [allocation_type]4096, [prote
<process>-<0,0x00000000> NtWriteVirtualMemory([process_handle]0x000002BC, [base_address]0x00070000, [process_identifier]7572)
<process>-<0,0x00000000> NtWriteVirtualMemory([process_handle]0x000002BC, [base_address]0x002311E8, [process_identifier]7572)
<system>-<0,0x00000000> NtClose([handle]0x000002C4)
<system>-<0,0x00000000> NtClose([handle]0x000002EC)
<process>-<0,0x00000000> NtGetContextThread([thread_handle]0x000002E8)
<process>-<1,0x00000001> ReadProcessMemory([process_handle]0x000002BC, [base_address]0x00231008)
<file>-<0,0x00000000> NtCreateFile([file_handle]0x000002EC, [desired_access]-2146434944, [file_attributes]128, [create_disposition]1, [create_options]96, [sh
<file>-<1678152,0x00199B48> GetFileSize([file_handle]0x000002EC, [file_size_low]1678152)
<process>-<0,0x00000000> NtAllocateVirtualMemory([process_handle]0xFFFFFFFF, [base_address]0x00BB0000, [region_size]0x0019A000, [allocation_type]12288, [prot
<file>-<0,0x00000000> NtReadFile([file_handle]0x000002EC, [length]1678152, [offset]0)
<process>-<0,0x00000000> NtAllocateVirtualMemory([process_handle]0xFFFFFFFF, [base_address]0x04A60000, [region_size]0x0019E000, [allocation_type]12288, [prot
<system>-<0,0x00000000> NtClose([handle]0x000002EC)
<process>-<0,0x00000000> NtFreeVirtualMemory([process_handle]0xFFFFFFFF, [base_address]0x00BB0000, [size]0x0019A000, [free_type]32768, [process_identifier]19
```



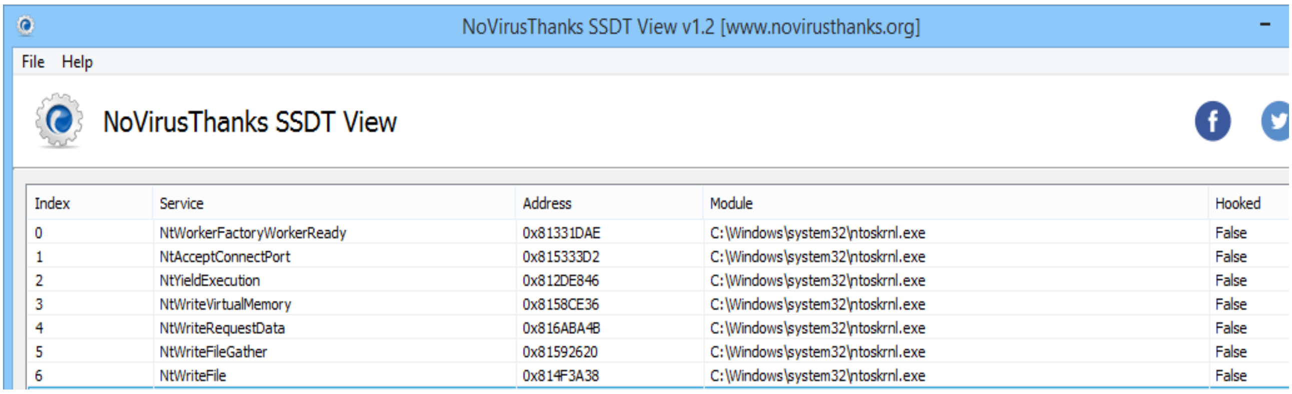
```
<process>-<0,0x00000000> NtAllocateVirtualMemory([process_handle]0xFFFFFFFF, [base_address]0x00BB0000, [region_size]0x0019A000, [allocation_type]12288, [pr
<file>-<0,0x00000000> NtReadFile([file_handle]0x000002EC, [length]1678152, [offset]0)
<process>-<0,0x00000000> NtAllocateVirtualMemory([process_handle]0xFFFFFFFF, [base_address]0x04A60000, [region_size]0x0019E000, [allocation_type]12288, [pr
[stack_pivoted]0, [stack_dep_bypass]0, [heap_dep_bypass]0, [process_identifier]1960)
<system>-<0,0x00000000> NtClose([handle]0x000002EC)
<process>-<0,0x00000000> NtFreeVirtualMemory([process_handle]0xFFFFFFFF, [base_address]0x00BB0000, [size]0x0019A000, [free_type]32768, [process_identifier]
<process>-<0,0x00000000> NtFreeVirtualMemory([process_handle]0xFFFFFFFF, [base_address]0x04A60000, [size]0x0019E000, [free_type]32768, [process_identifier]
<file>-<0,0x00000000> NtCreateFile([file_handle]0x000002C4, [desired_access]-2146434944, [file_attributes]128, [create_disposition]1,
[create_options]96, [share_access]7, [filepath]"C:\Windows\System32\ntdll.dll", [filepath_r]"\??\C:\Windows\SYSTEM32\ntdll.dll", [status_info]0x00000001)
<file>-<1678152,0x00199B48> GetFileSize([file_handle]0x000002C4, [file_size_low]1678152)
<process>-<0,0x00000000> NtAllocateVirtualMemory([process_handle]0xFFFFFFFF, [base_address]0x00BB0000, [region_size]0x0019A000, [allocation_type]12288, [pr
<file>-<0,0x00000000> NtReadFile([file_handle]0x000002C4, [length]1678152, [offset]0)
<process>-<0,0x00000000> NtAllocateVirtualMemory([process_handle]0xFFFFFFFF, [base_address]0x04A60000, [region_size]0x0019E000, [allocation_type]12288, [pr
```

So, we can see that the Malware uses CreateProcessW() and GetThreadContext(), SetThreadContext() functions, but there is no ntMapViewofSection() or ResumeThread() or other Api’s responsible for process injection. But we can see ntdll file being read which it will used to get Syscall numbers.

So what is actually going on? To know that we have to know how windows user to kernel mode interaction works.

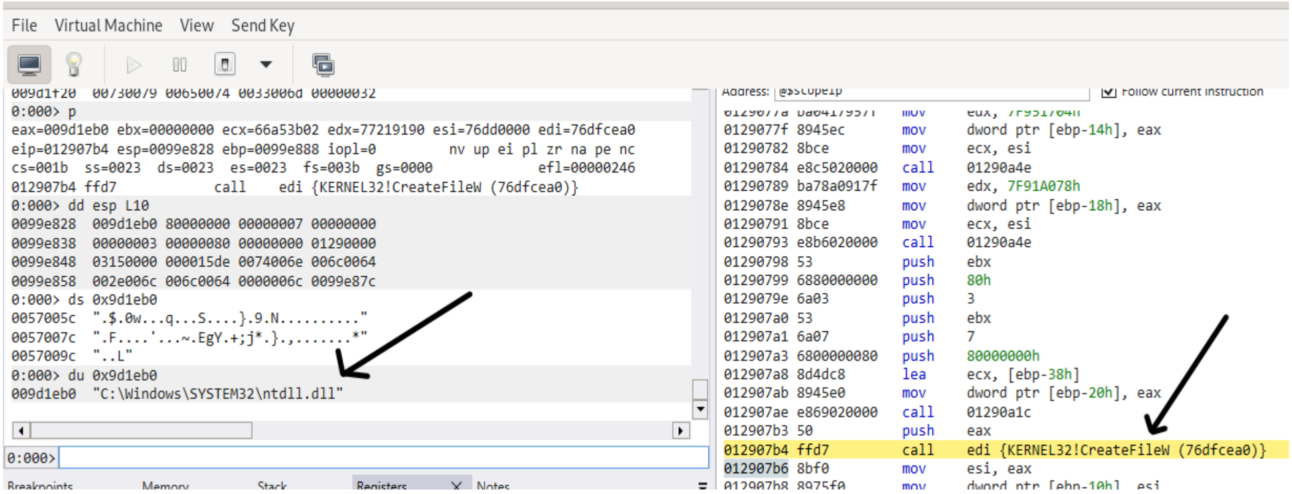


Normally a malware use  hasehes to find  the address of system calls in kernel32.dll and than calls that address. But loki malware gets the syscall ID number of the systemcall from ntdll.dll and uses that and the sysenter instruction to do its task. Whats a syscall ID? Syscall ID/number is basically the position in which the systemcall address resides in the System ServiceDescriptor Table(SSDT). SSDT table is what the kernel uses to find the address of the Service to be executed in kernel space. Below is an example of what an SSDT looks like.



| Index | Service | Address | Module | Hooked |
|---|---|---|---|---|
| 0 | NtWorkerFactoryWorkerReady | 0x81331DAE | C:\Windows\system32\ntoskrnl.exe | False |
| 1 | NtAcceptConnectPort | 0x815333D2 | C:\Windows\system32\ntoskrnl.exe | False |
| 2 | NtYieldExecution | 0x812DE846 | C:\Windows\system32\ntoskrnl.exe | False |
| 3 | NtWriteVirtualMemory | 0x8158CE36 | C:\Windows\system32\ntoskrnl.exe | False |
| 4 | NtWriteRequestData | 0x816ABA4B | C:\Windows\system32\ntoskrnl.exe | False |
| 5 | NtWriteFileGather | 0x81592620 | C:\Windows\system32\ntoskrnl.exe | False |
| 6 | NtWriteFile | 0x814F3A38 | C:\Windows\system32\ntoskrnl.exe | False |

The Malware first reads the ntdll.dll.

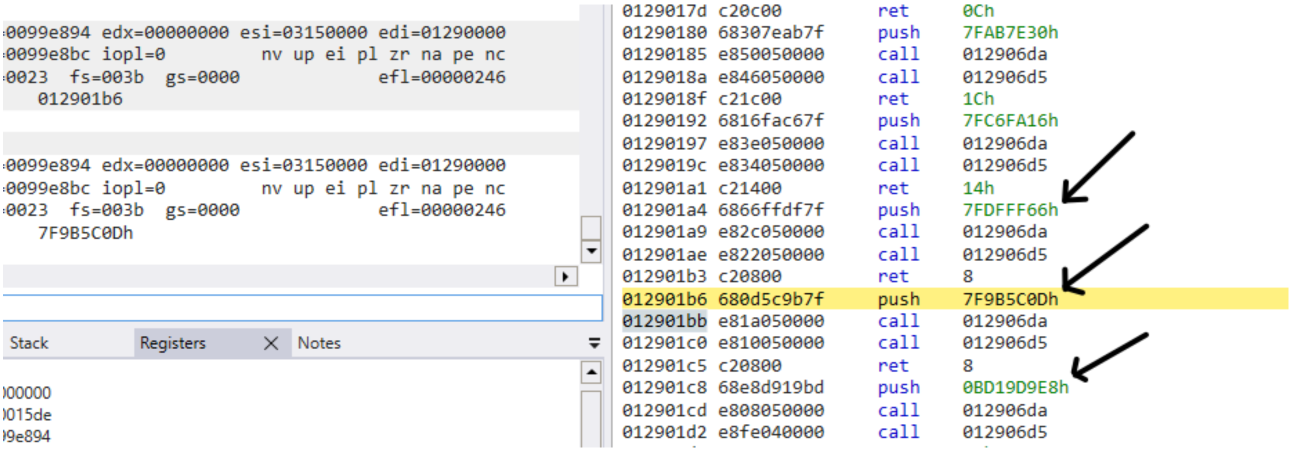


File Virtual Machine View Send Key
Address: @$scopeip
☑ Follow current instruction
Breakpoints Memory Stack Registers Notes

```
009d1f20  00730079 00650074 0033006d 00000032
0:000> p
eax=009d1eb0 ebx=00000000 ecx=66a53b02 edx=77219190 esi=76dd0000 edi=76dfcea0
eip=012907b4 esp=0099e828 ebp=0099e888 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
012907b4 ffd7            call    edi {KERNEL32!CreateFileW (76dfcea0)}
0:000> dd esp L10
0099e828  009d1eb0 80000000 00000007 00000000
0099e838  00000003 00000080 00000000 01290000
0099e848  03150000 000015de 0074006e 006c0064
0099e858  002e006c 006c0064 0000006c 0099e87c
0:000> ds 0x9d1eb0
0057005c  ".$.0w...q...S....}.9.N.........."
0057007c  ".F....'...~.EgY.+;j*.}.,.......*"
0057009c  "..L"
0:000> du 0x9d1eb0
009d1eb0  "C:\Windows\SYSTEM32\ntdll.dll"
0:000>
```

```
0129077f 8945ec      mov     dword ptr [ebp-14h], eax
01290782 8bce        mov     ecx, esi
01290784 e8c5020000  call    01290a4e
01290789 ba78a0917f  mov     edx, 7F91A078h
0129078e 8945e8      mov     dword ptr [ebp-18h], eax
01290791 8bce        mov     ecx, esi
01290793 e8b6020000  call    01290a4e
01290798 53          push    ebx
01290799 6880000000  push    80h
0129079e 6a03        push    3
012907a0 53          push    ebx
012907a1 6a07        push    7
012907a3 6800000080  push    80000000h
012907a8 8d4dc8      lea     ecx, [ebp-38h]
012907ab 8945e0      mov     dword ptr [ebp-20h], eax
012907ae e869020000  call    01290a1c
012907b3 50          push    eax
012907b4 ffd7        call    edi {KERNEL32!CreateFileW (76dfcea0)}
012907b6 8bf0        mov     esi, eax
012907b8 8975f0      mov     dword ptr [ebp-10h], esi
```

It then uses a hash to look for the required API in the ntdll file and get the starting address of the API.

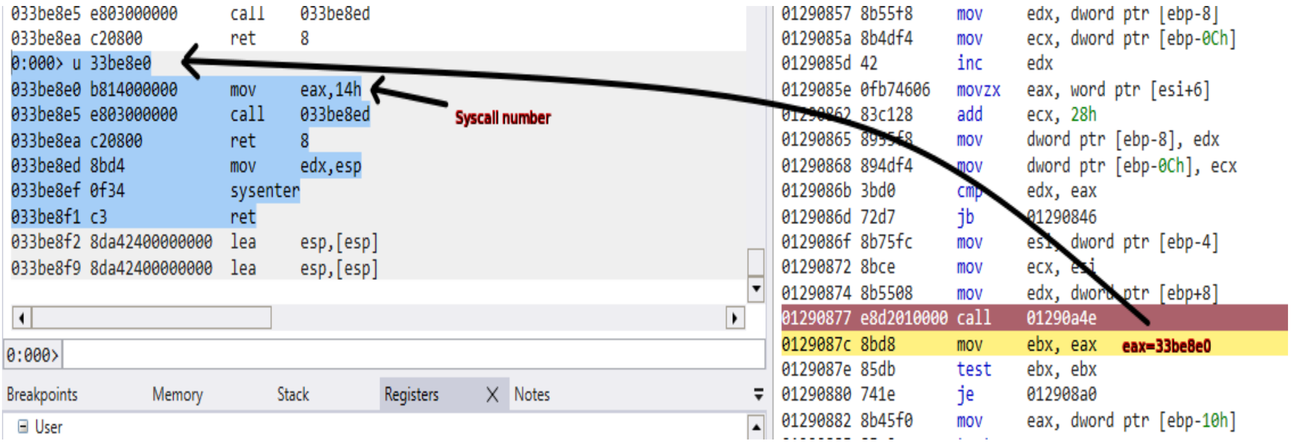


```
0099e894 edx=00000000 esi=03150000 edi=01290000
0099e8bc iopl=0         nv up ei pl zr na pe nc
0023  fs=003b  gs=0000             efl=00000246
  012901b6

0099e894 edx=00000000 esi=03150000 edi=01290000
0099e8bc iopl=0         nv up ei pl zr na pe nc
0023  fs=003b  gs=0000             efl=00000246
  7F9B5C0Dh

00000
015de
9e894
```

```
0129017d c20c00      ret     0Ch
01290180 68307eab7f  push    7FAB7E30h
01290185 e850050000  call    012906da
0129018a e846050000  call    012906d5
0129018f c21c00      ret     1Ch
01290192 6816fac67f  push    7FC6FA16h
01290197 e83e050000  call    012906da
0129019c e834050000  call    012906d5
012901a1 c21400      ret     14h
012901a4 6866ffdf7f  push    7FDFFF66h
012901a9 e82c050000  call    012906da
012901ae e822050000  call    012906d5
012901b3 c20800      ret     8
012901b6 680d5c9b7f  push    7F9B5C0Dh
012901bb e81a050000  call    012906da
012901c0 e810050000  call    012906d5
012901c5 c20800      ret     8
012901c8 68e8d919bd  push    0BD19D9E8h
012901cd e808050000  call    012906da
012901d2 e8fe040000  call    012906d5
```

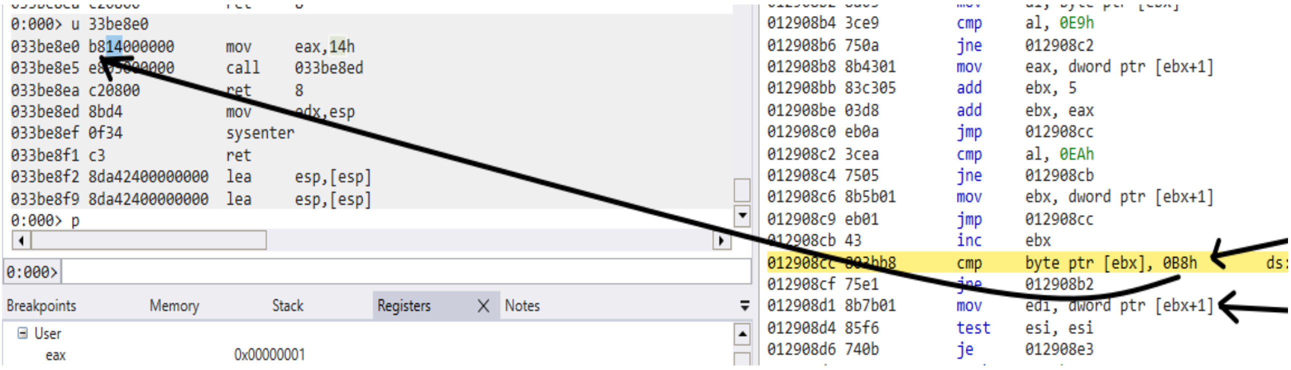


```
033be8e5 e803000000  call    033be8ed
033be8ea c20800      ret     8
0:000> u 33be8e0
033be8e0 b814000000  mov     eax,14h
033be8e5 e803000000  call    033be8ed
033be8ea c20800      ret     8
033be8ed 8bd4        mov     edx,esp
033be8ef 0f34        sysenter
033be8f1 c3          ret
033be8f2 8da42400000000 lea  esp,[esp]
033be8f9 8da42400000000 lea  esp,[esp]
0:000>
```

```
01290857 8b55f8      mov     edx, dword ptr [ebp-8]
0129085a 8b4df4      mov     ecx, dword ptr [ebp-0Ch]
0129085d 42          inc     edx
0129085e 0fb74606    movzx   eax, word ptr [esi+6]
01290862 83c128      add     ecx, 28h
01290865 8955f8      mov     dword ptr [ebp-8], edx
01290868 894df4      mov     dword ptr [ebp-0Ch], ecx
0129086b 3bd0        cmp     edx, eax
0129086d 72d7        jb      01290846
0129086f 8b75fc      mov     esi, dword ptr [ebp-4]
01290872 8bce        mov     ecx, esi
01290874 8b5508      mov     edx, dword ptr [ebp+8]
01290877 e8d2010000  call    01290a4e
0129087c 8bd8        mov     ebx, eax
0129087e 85db        test    ebx, ebx
01290880 741e        je      012908a0
01290882 8b45f0      mov     eax, dword ptr [ebp-10h]
```

The function call at address 0x1290877 searches for the addres of the function require. Syscall ID 0x14 in this case maps to ntMapViewOfSection.

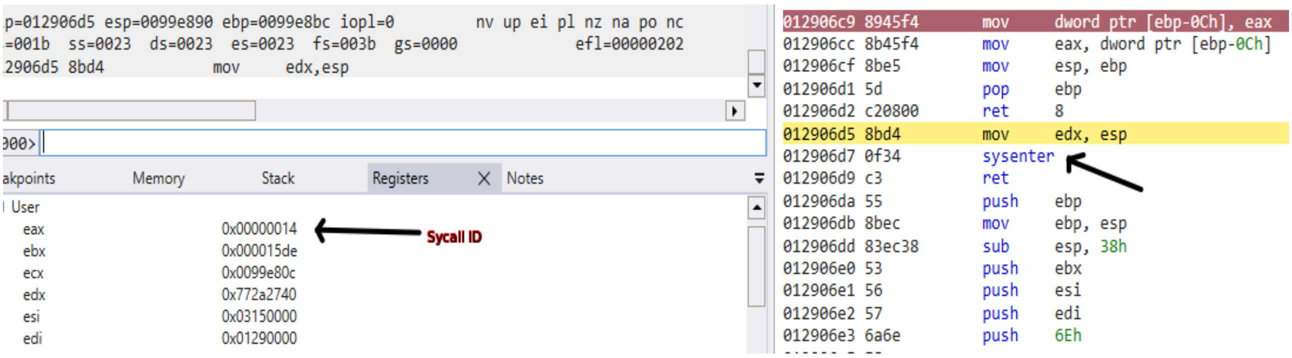
Then it check for the first byte of the instruction “mov eax, X” if its “B8” which corresponds to “mov eax” then it saves the ordinal number represented by the “X” and saves it in eax and uses the sysenter instuction to execute the system service.



```
0:000> u 33be8e0
033be8e0 b814000000       mov     eax,14h
033be8e5 e803000000       call    033be8ed
033be8ea c20800           ret     8
033be8ed 8bd4             mov     edx,esp
033be8ef 0f34             sysenter
033be8f1 c3               ret
033be8f2 8da42400000000   lea     esp,[esp]
033be8f9 8da42400000000   lea     esp,[esp]
0:000> p
```

```
eax                0x00000001
```

```
012908b4 3ce9       cmp    al, 0E9h
012908b6 750a       jne    012908c2
012908b8 8b4301     mov    eax, dword ptr [ebx+1]
012908bb 83c305     add    ebx, 5
012908be 03d8       add    ebx, eax
012908c0 eb0a       jmp    012908cc
012908c2 3cea       cmp    al, 0EAh
012908c4 7505       jne    012908cb
012908c6 8b5b01     mov    ebx, dword ptr [ebx+1]
012908c9 eb01       jmp    012908cc
012908cb 43         inc    ebx
012908cc 803bb8     cmp    byte ptr [ebx], 0B8h        ds:
012908cf 75e1       jne    012908b2
012908d1 8b7b01     mov    edi, dword ptr [ebx+1]
012908d4 85f6       test   esi, esi
012908d6 740b       je     012908e3
```



```
p=012906d5 esp=0099e890 ebp=0099e8bc iopl=0         nv up ei pl nz na po nc
=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000202
2906d5 8bd4            mov     edx,esp
```

```
eax                0x00000014
ebx                0x000015de
ecx                0x0099e80c
edx                0x772a2740
esi                0x03150000
edi                0x01290000
```

```
012906c9 8945f4     mov    dword ptr [ebp-0Ch], eax
012906cc 8b45f4     mov    eax, dword ptr [ebp-0Ch]
012906cf 8be5       mov    esp, ebp
012906d1 5d         pop    ebp
012906d2 c20800     ret    8
012906d5 8bd4       mov    edx, esp
012906d7 0f34       sysenter
012906d9 c3         ret
012906da 55         push   ebp
012906db 8bec       mov    ebp, esp
012906dd 83ec38     sub    esp, 38h
012906e0 53         push   ebx
012906e1 56         push   esi
012906e2 57         push   edi
012906e3 6a6e       push   6Eh
```

Conslusion :- This is not a significant change in the behaviour of malware but shows simple changes being used for evasion