Topic :- How malware resolve API addresses dynamically

Malware hashes :-

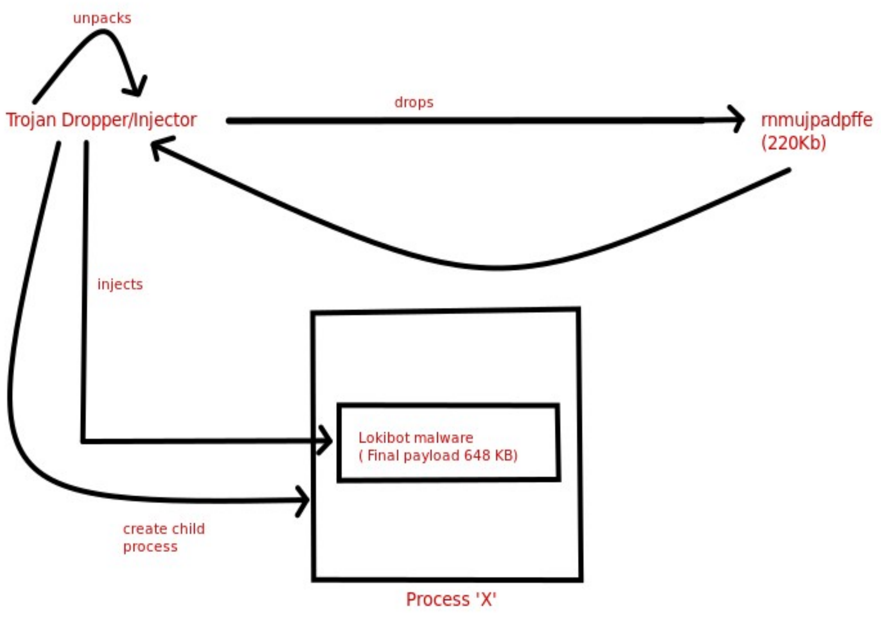
- **sha1** :- 70c34a5e1442816c23d78454edc2c7505f43f82b
- **sha256** :- 563818872af4977ebccd2bc8f97e968edeb6cce444c7a380b3c69e53fd317c2e

Tools Used :- Windbg

Overview :- Malware often try to hide their intentions and one simple trick is to hide the API's they use which can in most cases reveal their intention. They achieve this using Dynamic API address resolution which we will discuss in this article.

Whenever a malware analyst gets a malware file most of the time the first steps taken is to see the strings and imported API's by the malware. But this is hindered when the malware is packed and sometimes even after unpacking the malware we can see the strings but not always the imported API's. Which then requires the Analyst to resolve to Dynamic Analysis or code reversing of malware code. So, in this Article, I would like to present a clear picture as to how the malware goes about resolving API address dynamically using the Lokibot Malware sample.

So below image shows the injector part of the Loki Malware.



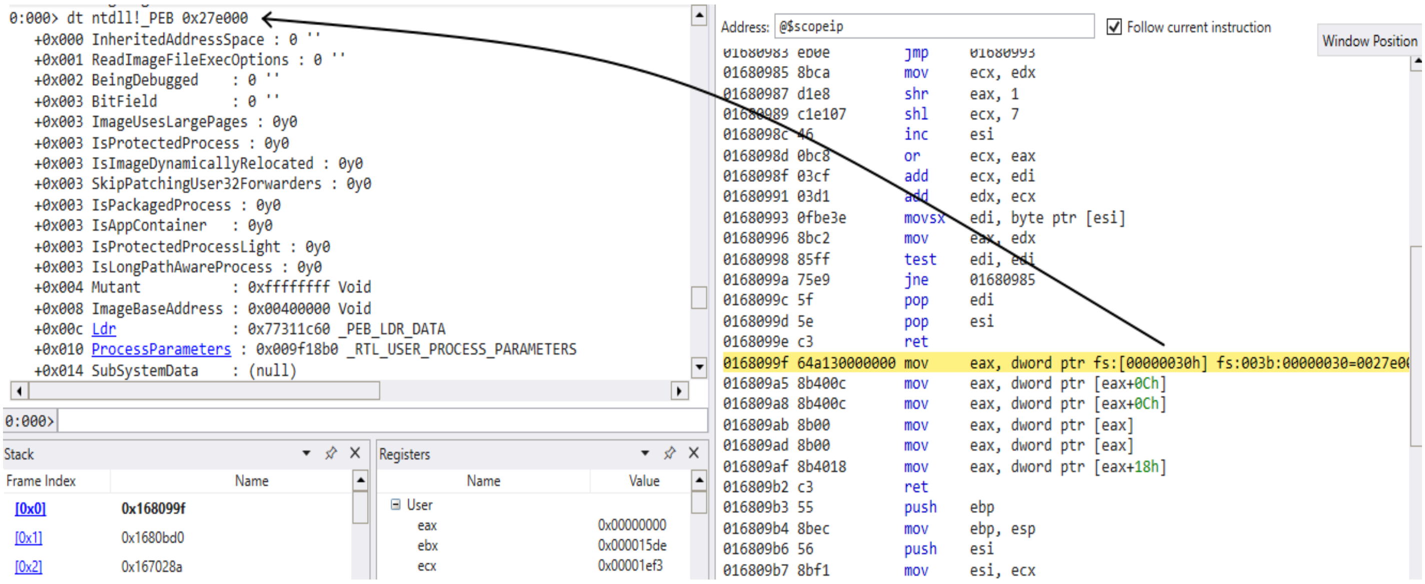
In order to achieve this task it has to use windows API's or system call interface which it can do in many ways. Some of the techniques are.

- It uses kernel32.dll to get the API address it wants.
- It uses the ntdll.dll to get the Syscall ID and uses sysenter to achieve the functionalty required.

In this article we will focus on the first. I have already discussed the second technique in depth in a previous artcile which you can check out *here*.

So we can see from the above image that it creates a new process from its own binary image. In order to do that it uses the particular windows API, in this case CreateProcessW. So lets see how it gets the address of that API.

It first gets the address of the PEB(Process Environment Block) at line 0x0168099f. Then it goes on to access the PEB_LDR_DATA structure field at line 0x016809a5. Then inside PEB_LDR_DATA structure it accesses the InLoadOrderModuleList structure. It points to a *Doubly linked* list. This list holds a structure of type LDR_DATA_TABLE_ENTRY.



```
0:000> dt ntdll!_PEB 0x27e000
   +0x000 InheritedAddressSpace : 0 ''
   +0x001 ReadImageFileExecOptions : 0 ''
   +0x002 BeingDebugged    : 0 ''
   +0x003 BitField         : 0 ''
   +0x003 ImageUsesLargePages : 0y0
   +0x003 IsProtectedProcess : 0y0
   +0x003 IsImageDynamicallyRelocated : 0y0
   +0x003 SkipPatchingUser32Forwarders : 0y0
   +0x003 IsPackagedProcess : 0y0
   +0x003 IsAppContainer   : 0y0
   +0x003 IsProtectedProcessLight : 0y0
   +0x003 IsLongPathAwareProcess : 0y0
   +0x004 Mutant           : 0xffffffff Void
   +0x008 ImageBaseAddress : 0x00400000 Void
   +0x00c Ldr              : 0x77311c60 _PEB_LDR_DATA
   +0x010 ProcessParameters : 0x009f18b0 _RTL_USER_PROCESS_PARAMETERS
   +0x014 SubSystemData    : (null)
0:000>
```

Stack

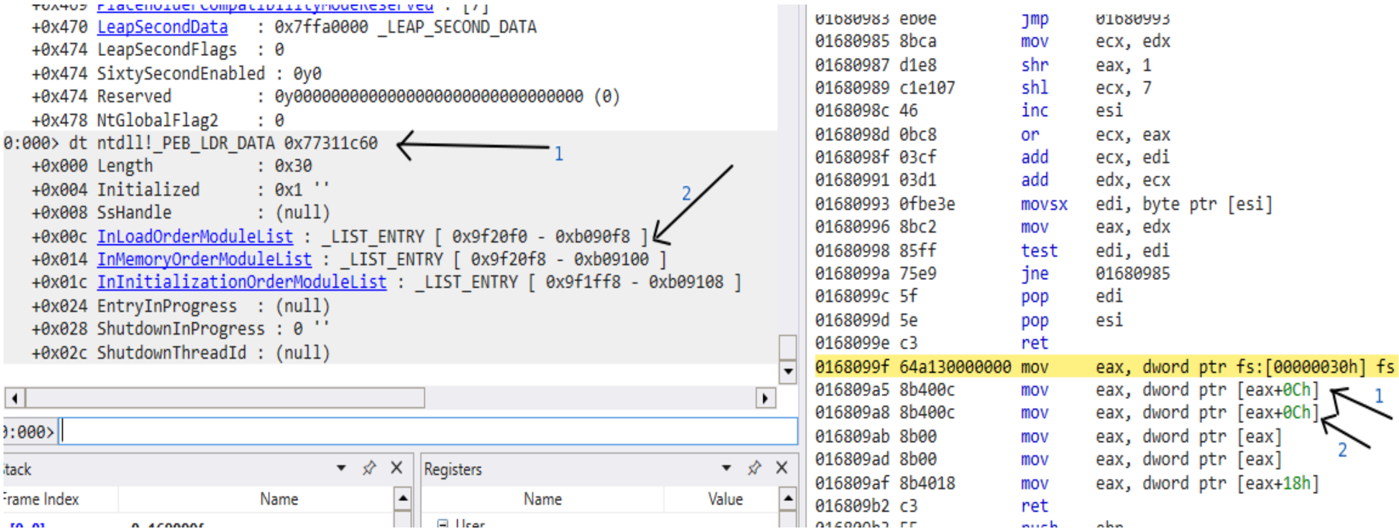
Frame Index	Name
[0x0]	**0x168099f**
[0x1]	0x1680bd0
[0x2]	0x167028a

Registers

Name	Value
⊟ User	
eax	0x00000000
ebx	0x000015de
ecx	0x00001ef3

Address: @$scopeip ☑ Follow current instruction

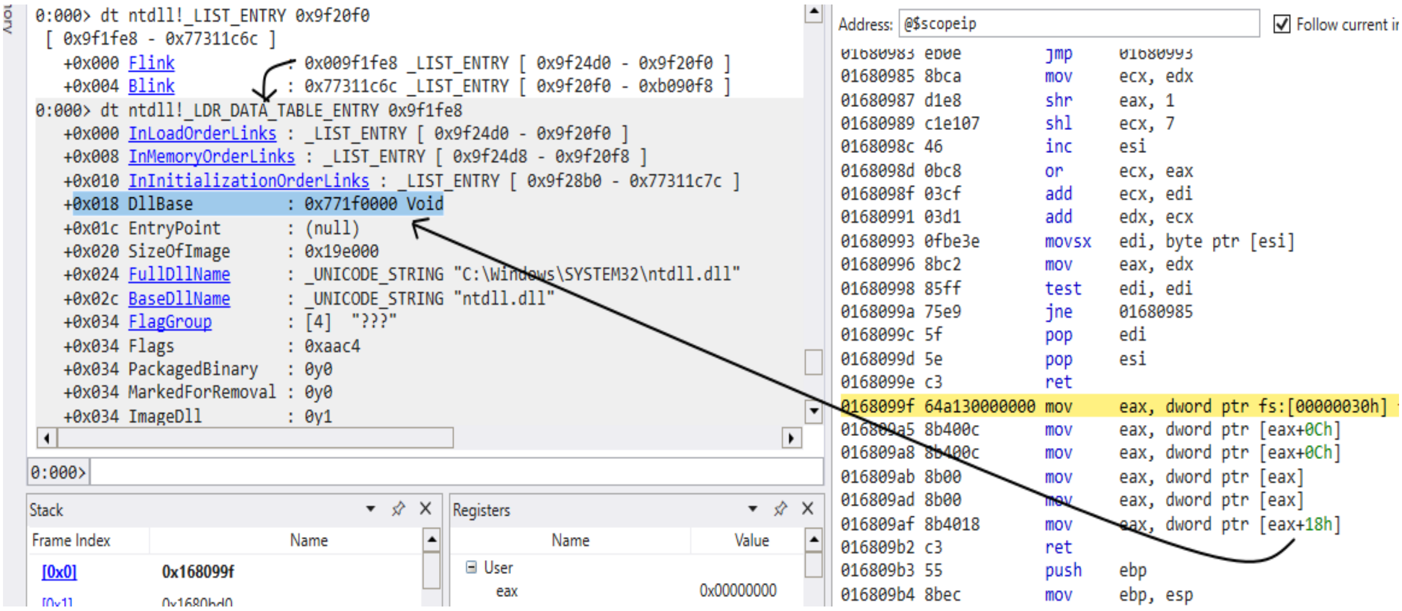
```
01680983 eb0e         jmp     01680993
01680985 8bca         mov     ecx, edx
01680987 d1e8         shr     eax, 1
01680989 c1e107       shl     ecx, 7
0168098c 46           inc     esi
0168098d 0bc8         or      ecx, eax
0168098f 03cf         add     ecx, edi
01680991 03d1         add     edx, ecx
01680993 0fbe3e       movsx   edi, byte ptr [esi]
01680996 8bc2         mov     eax, edx
01680998 85ff         test    edi, edi
0168099a 75e9         jne     01680985
0168099c 5f           pop     edi
0168099d 5e           pop     esi
0168099e c3           ret
0168099f 64a130000000 mov     eax, dword ptr fs:[00000030h] fs:003b:00000030=0027e0
016809a5 8b400c       mov     eax, dword ptr [eax+0Ch]
016809a8 8b400c       mov     eax, dword ptr [eax+0Ch]
016809ab 8b00         mov     eax, dword ptr [eax]
016809ad 8b00         mov     eax, dword ptr [eax]
016809af 8b4018       mov     eax, dword ptr [eax+18h]
016809b2 c3           ret
016809b3 55           push    ebp
016809b4 8bec         mov     ebp, esp
016809b6 56           push    esi
016809b7 8bf1         mov     esi, ecx
```



```
   +0x469 PlaceholderCompatibilityModeReserved : [7]
   +0x470 LeapSecondData   : 0x7ffa0000 _LEAP_SECOND_DATA
   +0x474 LeapSecondFlags  : 0
   +0x474 SixtySecondEnabled : 0y0
   +0x474 Reserved         : 0y0000000000000000000000000000000 (0)
   +0x478 NtGlobalFlag2    : 0
0:000> dt ntdll!_PEB_LDR_DATA 0x77311c60
   +0x000 Length           : 0x30
   +0x004 Initialized      : 0x1 ''
   +0x008 SsHandle         : (null)
   +0x00c InLoadOrderModuleList : _LIST_ENTRY [ 0x9f20f0 - 0xb090f8 ]
   +0x014 InMemoryOrderModuleList : _LIST_ENTRY [ 0x9f20f8 - 0xb09100 ]
   +0x01c InInitializationOrderModuleList : _LIST_ENTRY [ 0x9f1ff8 - 0xb09108 ]
   +0x024 EntryInProgress  : (null)
   +0x028 ShutdownInProgress : 0 ''
   +0x02c ShutdownThreadId : (null)
0:000>
```

```
01680983 eb0e         jmp     01680993
01680985 8bca         mov     ecx, edx
01680987 d1e8         shr     eax, 1
01680989 c1e107       shl     ecx, 7
0168098c 46           inc     esi
0168098d 0bc8         or      ecx, eax
0168098f 03cf         add     ecx, edi
01680991 03d1         add     edx, ecx
01680993 0fbe3e       movsx   edi, byte ptr [esi]
01680996 8bc2         mov     eax, edx
01680998 85ff         test    edi, edi
0168099a 75e9         jne     01680985
0168099c 5f           pop     edi
0168099d 5e           pop     esi
0168099e c3           ret
0168099f 64a130000000 mov     eax, dword ptr fs:[00000030h] fs
016809a5 8b400c       mov     eax, dword ptr [eax+0Ch]
016809a8 8b400c       mov     eax, dword ptr [eax+0Ch]
016809ab 8b00         mov     eax, dword ptr [eax]
016809ad 8b00         mov     eax, dword ptr [eax]
016809af 8b4018       mov     eax, dword ptr [eax+18h]
016809b2 c3           ret
```

The LDR_DATA_TABLE_ENTRY structure contains various field as shown below. At offset 0x18 it contains the base of the dll, which is accessed at line 0x016809af.



```
0:000> dt ntdll!_LIST_ENTRY 0x9f20f0
 [ 0x9f1fe8 - 0x77311c6c ]
   +0x000 Flink            : 0x009f1fe8 _LIST_ENTRY [ 0x9f24d0 - 0x9f20f0 ]
   +0x004 Blink            : 0x77311c6c _LIST_ENTRY [ 0x9f20f0 - 0xb090f8 ]
0:000> dt ntdll!_LDR_DATA_TABLE_ENTRY 0x9f1fe8
   +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x9f24d0 - 0x9f20f0 ]
   +0x008 InMemoryOrderLinks : _LIST_ENTRY [ 0x9f24d8 - 0x9f20f8 ]
   +0x010 InInitializationOrderLinks : _LIST_ENTRY [ 0x9f28b0 - 0x77311c7c ]
   +0x018 DllBase          : 0x771f0000 Void
   +0x01c EntryPoint       : (null)
   +0x020 SizeOfImage      : 0x19e000
   +0x024 FullDllName      : _UNICODE_STRING "C:\Windows\SYSTEM32\ntdll.dll"
   +0x02c BaseDllName      : _UNICODE_STRING "ntdll.dll"
   +0x034 FlagGroup        : [4]  "???"
   +0x034 Flags            : 0xaac4
   +0x034 PackagedBinary   : 0y0
   +0x034 MarkedForRemoval : 0y0
   +0x034 ImageDll         : 0y1
0:000>
```

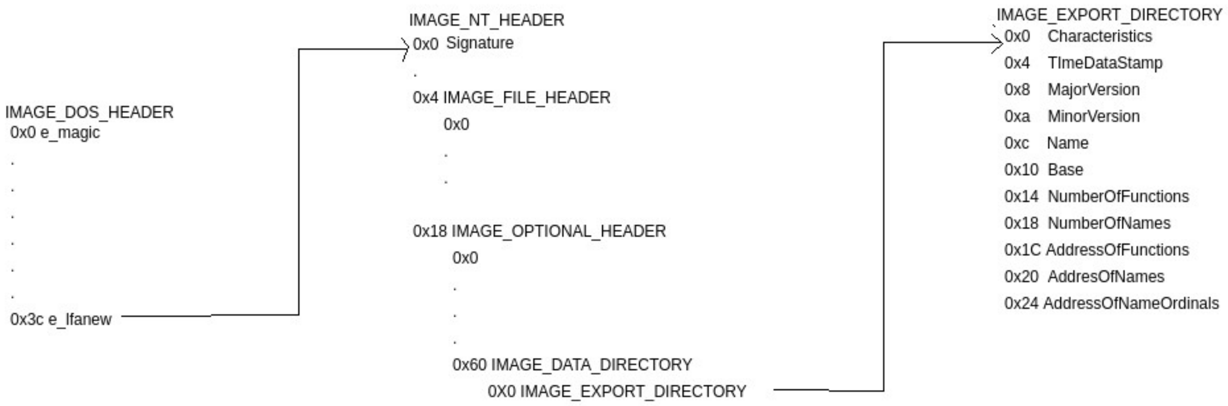
Frame Index	Name
[0x0]	0x168099f
[0x1]	0x1680bd0

Name	Value
⊟ User	
eax	0x00000000

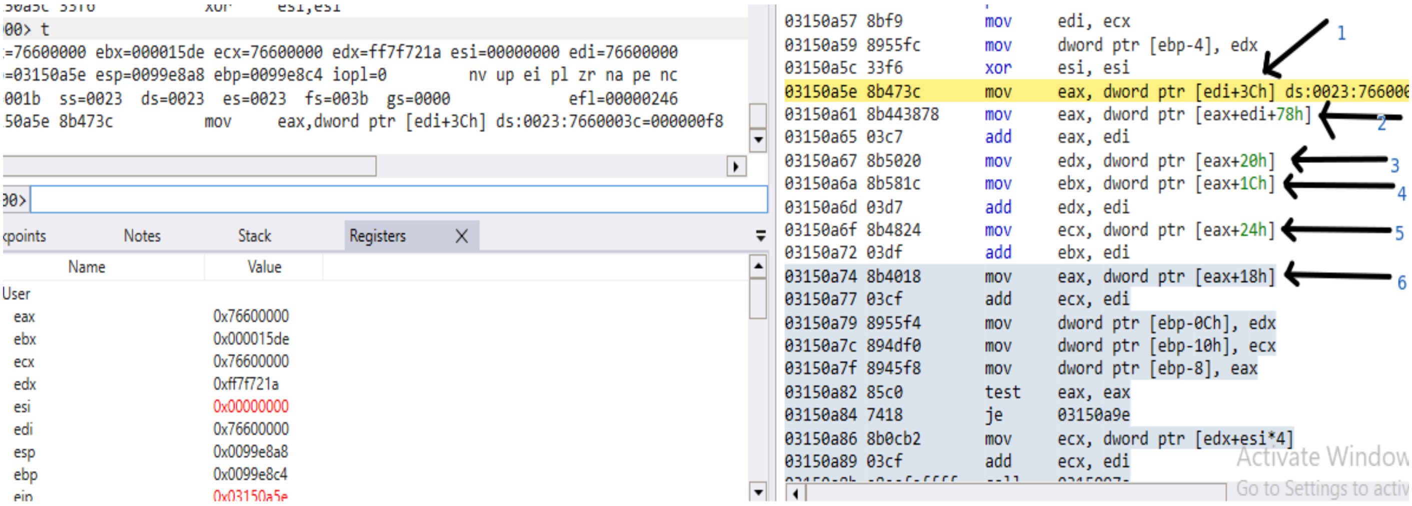
Address: @$scopeip ☑ Follow current ir

```
01680983 eb0e         jmp     01680993
01680985 8bca         mov     ecx, edx
01680987 d1e8         shr     eax, 1
01680989 c1e107       shl     ecx, 7
0168098c 46           inc     esi
0168098d 0bc8         or      ecx, eax
0168098f 03cf         add     ecx, edi
01680991 03d1         add     edx, ecx
01680993 0fbe3e       movsx   edi, byte ptr [esi]
01680996 8bc2         mov     eax, edx
01680998 85ff         test    edi, edi
0168099a 75e9         jne     01680985
0168099c 5f           pop     edi
0168099d 5e           pop     esi
0168099e c3           ret
0168099f 64a130000000 mov     eax, dword ptr fs:[00000030h]
016809a5 8b400c       mov     eax, dword ptr [eax+0Ch]
016809a8 8b400c       mov     eax, dword ptr [eax+0Ch]
016809ab 8b00         mov     eax, dword ptr [eax]
016809ad 8b00         mov     eax, dword ptr [eax]
016809af 8b4018       mov     eax, dword ptr [eax+18h]
016809b2 c3           ret
016809b3 55           push    ebp
016809b4 8bec         mov     ebp, esp
```

After getting the base address of the kernel32.dll, the malware traverses the dll to get the function address. In order to understand that we need to know the layout of the PE(Portable Executable) file format. Below is a high level overview of the PE format.



Below we can see how the malware implements the functionality of getting function address.



```
50a5c 33f6          xor     esi,esi
00> t
=76600000 ebx=000015de ecx=76600000 edx=ff7f721a esi=00000000 edi=76600000
=03150a5e esp=0099e8a8 ebp=0099e8c4 iopl=0         nv up ei pl zr na pe nc
001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
50a5e 8b473c          mov     eax,dword ptr [edi+3Ch] ds:0023:7660003c=000000f8
```

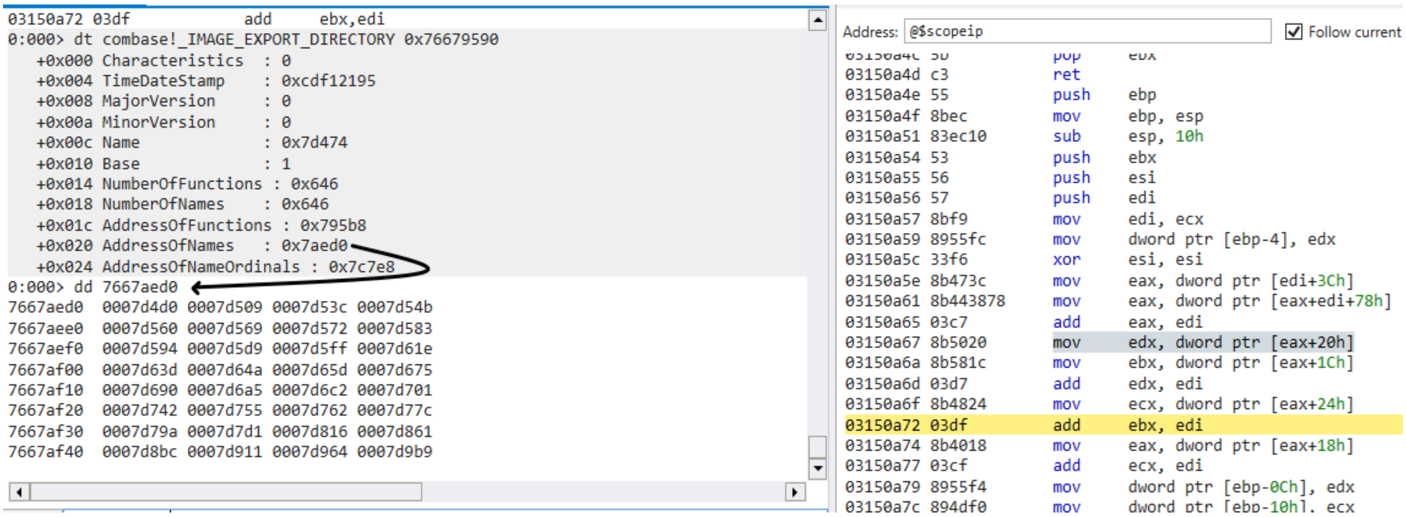
```
Name            Value
User
  eax           0x76600000
  ebx           0x000015de
  ecx           0x76600000
  edx           0xff7f721a
  esi           0x00000000
  edi           0x76600000
  esp           0x0099e8a8
  ebp           0x0099e8c4
  eip           0x03150a5e
```

```
03150a57 8bf9        mov     edi, ecx
03150a59 8955fc      mov     dword ptr [ebp-4], edx
03150a5c 33f6        xor     esi, esi
03150a5e 8b473c      mov     eax, dword ptr [edi+3Ch] ds:0023:76600
03150a61 8b443878    mov     eax, dword ptr [eax+edi+78h]
03150a65 03c7        add     eax, edi
03150a67 8b5020      mov     edx, dword ptr [eax+20h]
03150a6a 8b581c      mov     ebx, dword ptr [eax+1Ch]
03150a6d 03d7        add     edx, edi
03150a6f 8b4824      mov     ecx, dword ptr [eax+24h]
03150a72 03df        add     ebx, edi
03150a74 8b4018      mov     eax, dword ptr [eax+18h]
03150a77 03cf        add     ecx, edi
03150a79 8955f4      mov     dword ptr [ebp-0Ch], edx
03150a7c 894df0      mov     dword ptr [ebp-10h], ecx
03150a7f 8945f8      mov     dword ptr [ebp-8], eax
03150a82 85c0        test    eax, eax
03150a84 7418        je      03150a9e
03150a86 8b0cb2      mov     ecx, dword ptr [edx+esi*4]
03150a89 03cf        add     ecx, edi
```

It first accesses the e_lfanew field of IMAGE_DOS_HEADER which points to IMAGE_NT_HEADER. Then at line 0x3150a61 it access the IMAGE_EXPORT_DIRECTORY Field. After getting the base of IMAGE_EXPORT_DIRECTORY structure, it access the fields in the Structure as shown above in the image.

- IMAGE_EXPORT_DIRECTORY + 0x20 = AddressOfNames
- IMAGE_EXPORT_DIRECTORY + 0x1C = AddressOfFunctions
- IMAGE_EXPORT_DIRECTORY + 0x24 = AddressOfNameOrdinals
- IMAGE_EXPORT_DIRECTORY + 0x18 = NumberOfNames

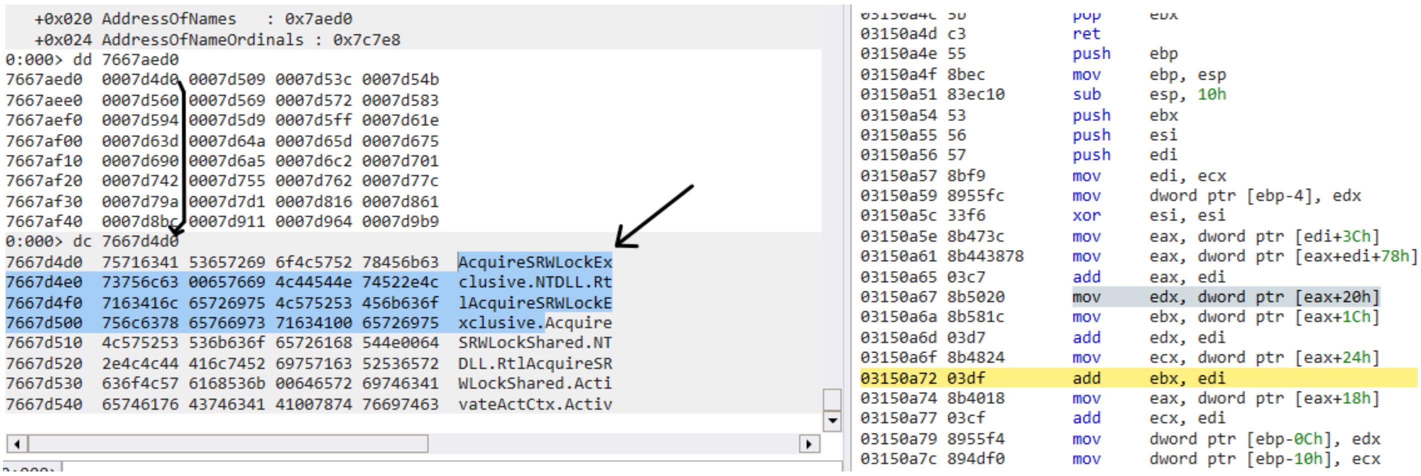
The AddressOfNames field points to an array of pointers, where each address points to the name of a particular function.



```
03150a72 03df            add     ebx,edi
0:000> dt combase!_IMAGE_EXPORT_DIRECTORY 0x76679590
   +0x000 Characteristics  : 0
   +0x004 TimeDateStamp    : 0xcdf12195
   +0x008 MajorVersion     : 0
   +0x00a MinorVersion     : 0
   +0x00c Name             : 0x7d474
   +0x010 Base             : 1
   +0x014 NumberOfFunctions : 0x646
   +0x018 NumberOfNames    : 0x646
   +0x01c AddressOfFunctions : 0x795b8
   +0x020 AddressOfNames   : 0x7aed0
   +0x024 AddressOfNameOrdinals : 0x7c7e8
0:000> dd 7667aed0
7667aed0  0007d4d0 0007d509 0007d53c 0007d54b
7667aee0  0007d560 0007d569 0007d572 0007d583
7667aef0  0007d594 0007d5d9 0007d5ff 0007d61e
7667af00  0007d63d 0007d64a 0007d65d 0007d675
7667af10  0007d690 0007d6a5 0007d6c2 0007d701
7667af20  0007d742 0007d755 0007d762 0007d77c
7667af30  0007d79a 0007d7d1 0007d816 0007d861
7667af40  0007d8bc 0007d911 0007d964 0007d9b9
```

Address: @$scopeip ☑ Follow current

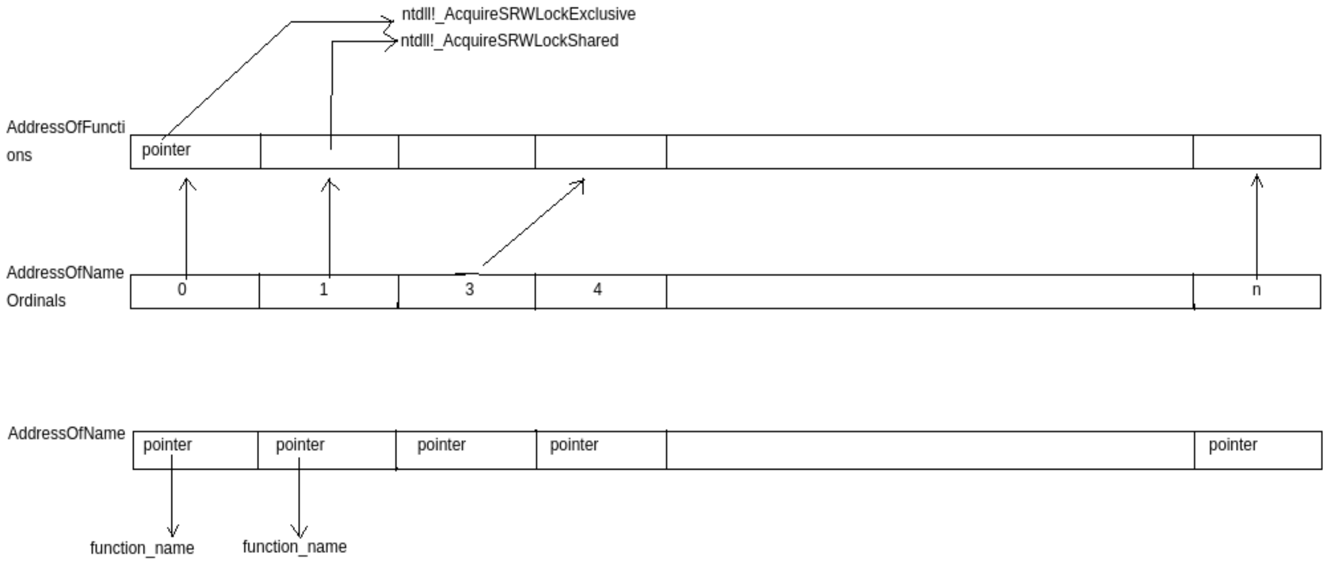
```
03150a4c 5b          pop     ebx
03150a4d c3          ret
03150a4e 55          push    ebp
03150a4f 8bec        mov     ebp, esp
03150a51 83ec10      sub     esp, 10h
03150a54 53          push    ebx
03150a55 56          push    esi
03150a56 57          push    edi
03150a57 8bf9        mov     edi, ecx
03150a59 8955fc      mov     dword ptr [ebp-4], edx
03150a5c 33f6        xor     esi, esi
03150a5e 8b473c      mov     eax, dword ptr [edi+3Ch]
03150a61 8b443878    mov     eax, dword ptr [eax+edi+78h]
03150a65 03c7        add     eax, edi
03150a67 8b5020      mov     edx, dword ptr [eax+20h]
03150a6a 8b581c      mov     ebx, dword ptr [eax+1Ch]
03150a6d 03d7        add     edx, edi
03150a6f 8b4824      mov     ecx, dword ptr [eax+24h]
03150a72 03df        add     ebx, edi
03150a74 8b4018      mov     eax, dword ptr [eax+18h]
03150a77 03cf        add     ecx, edi
03150a79 8955f4      mov     dword ptr [ebp-0Ch], edx
03150a7c 894df0      mov     dword ptr [ebp-10h], ecx
```



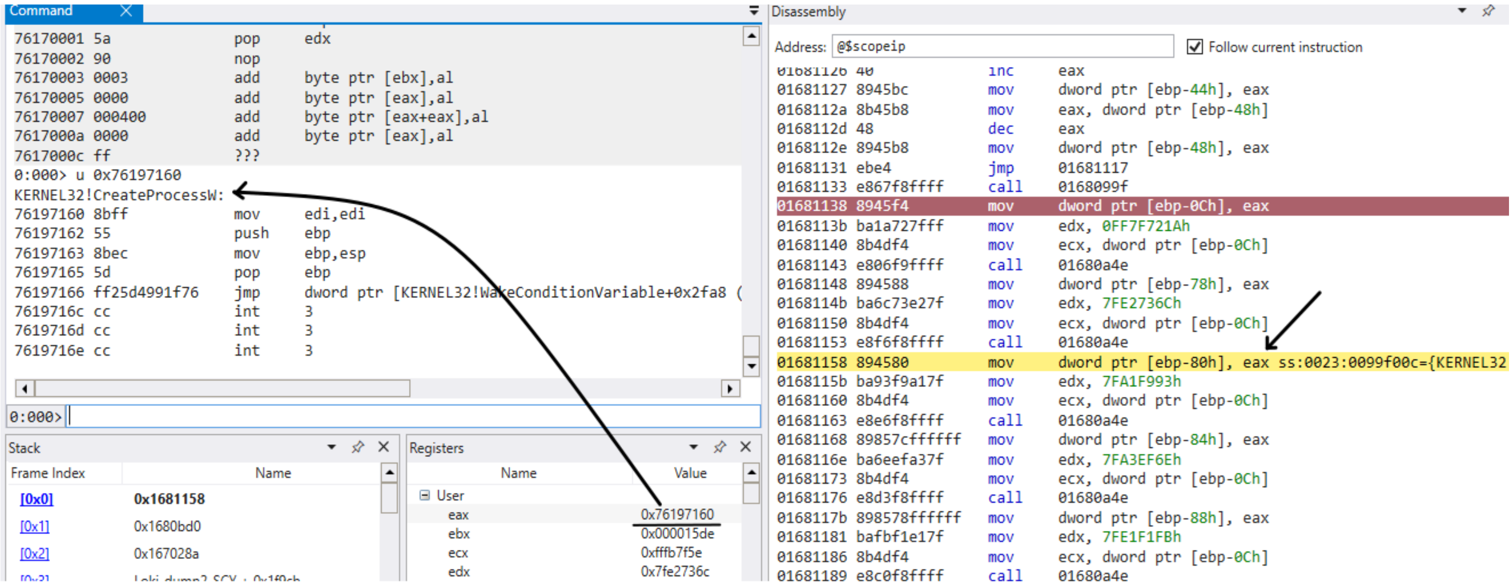
```
   +0x020 AddressOfNames    : 0x7aed0
   +0x024 AddressOfNameOrdinals : 0x7c7e8
0:000> dd 7667aed0
7667aed0  0007d4d0 0007d509 0007d53c 0007d54b
7667aee0  0007d560 0007d569 0007d572 0007d583
7667aef0  0007d594 0007d5d9 0007d5ff 0007d61e
7667af00  0007d63d 0007d64a 0007d65d 0007d675
7667af10  0007d690 0007d6a5 0007d6c2 0007d701
7667af20  0007d742 0007d755 0007d762 0007d77c
7667af30  0007d79a 0007d7d1 0007d816 0007d861
7667af40  0007d8bc 0007d911 0007d964 0007d9b9
0:000> dc 7667d4d0
7667d4d0  75716341 53657269 6f4c5752 78456b63  AcquireSRWLockEx
7667d4e0  73756c63 00657669 4c44544e 74522e4c  clusive.NTDLL.Rt
7667d4f0  7163416c 65726975 4c575253 456b636f  lAcquireSRWLockE
7667d500  756c6378 65766973 71634100 65726975  xclusive.Acquire
7667d510  4c575253 536b636f 65726168 544e0064  SRWLockShared.NT
7667d520  2e4c4c44 416c7452 69757163 52536572  DLL.RtlAcquireSR
7667d530  636f4c57 6168536b 00646572 69746341  WLockShared.Acti
7667d540  65746176 43746341 41007874 76697463  vateActCtx.Activ
```

```
03150a4c 5b             pop     ebx
03150a4d c3             ret
03150a4e 55             push    ebp
03150a4f 8bec           mov     ebp, esp
03150a51 83ec10         sub     esp, 10h
03150a54 53             push    ebx
03150a55 56             push    esi
03150a56 57             push    edi
03150a57 8bf9           mov     edi, ecx
03150a59 8955fc         mov     dword ptr [ebp-4], edx
03150a5c 33f6           xor     esi, esi
03150a5e 8b473c         mov     eax, dword ptr [edi+3Ch]
03150a61 8b443878       mov     eax, dword ptr [eax+edi+78h]
03150a65 03c7           add     eax, edi
03150a67 8b5020         mov     edx, dword ptr [eax+20h]
03150a6a 8b581c         mov     ebx, dword ptr [eax+1Ch]
03150a6d 03d7           add     edx, edi
03150a6f 8b4824         mov     ecx, dword ptr [eax+24h]
03150a72 03df           add     ebx, edi
03150a74 8b4018         mov     eax, dword ptr [eax+18h]
03150a77 03cf           add     ecx, edi
03150a79 8955f4         mov     dword ptr [ebp-0Ch], edx
03150a7c 894df0         mov     dword ptr [ebp-10h], ecx
```

The malware tries to iterate through all the names of APIs to find the matching API it requires. Sometimes it will directly match the API name with a predefined string and some times it calculate's a *Hash* of the API name and matche's that with a predefind *Hash* . The offset at which the function name is found, the same offset is used to access the AddressOfNameOrdinals array. The AddressOfNamesOrdinals field also points to an array of number. These number are the mapping from AddressOfNames to AddressOfFunctions array.



The Address obtained at the end is used to call the particular function.



Command

```
76170001 5a              pop     edx
76170002 90              nop
76170003 0003            add     byte ptr [ebx],al
76170005 0000            add     byte ptr [eax],al
76170007 000400          add     byte ptr [eax+eax],al
7617000a 0000            add     byte ptr [eax],al
7617000c ff              ???
0:000> u 0x76197160
KERNEL32!CreateProcessW:
76197160 8bff            mov     edi,edi
76197162 55              push    ebp
76197163 8bec            mov     ebp,esp
76197165 5d              pop     ebp
76197166 ff25d4991f76    jmp     dword ptr [KERNEL32!WakeConditionVariable+0x2fa8 (
7619716c cc              int     3
7619716d cc              int     3
7619716e cc              int     3
```

```
0:000>
```

Stack

Frame Index	Name
[0x0]	**0x1681158**
[0x1]	0x1680bd0
[0x2]	0x167028a
[0x3]	Loki_dump2.SCY + 0x1f0cb

Registers

Name	Value
⊟ User	
eax	0x76197160
ebx	0x000015de
ecx	0xfffb7f5e
edx	0x7fe2736c

Disassembly

Address: @$scopeip ☑ Follow current instruction

```
01681126 40              inc     eax
01681127 8945bc          mov     dword ptr [ebp-44h], eax
0168112a 8b45b8          mov     eax, dword ptr [ebp-48h]
0168112d 48              dec     eax
0168112e 8945b8          mov     dword ptr [ebp-48h], eax
01681131 ebe4            jmp     01681117
01681133 e867f8ffff      call    0168099f
01681138 8945f4          mov     dword ptr [ebp-0Ch], eax
0168113b ba1a727fff      mov     edx, 0FF7F721Ah
01681140 8b4df4          mov     ecx, dword ptr [ebp-0Ch]
01681143 e806f9ffff      call    01680a4e
01681148 894588          mov     dword ptr [ebp-78h], eax
0168114b ba6c73e27f      mov     edx, 7FE2736Ch
01681150 8b4df4          mov     ecx, dword ptr [ebp-0Ch]
01681153 e8f6f8ffff      call    01680a4e
01681158 894580          mov     dword ptr [ebp-80h], eax ss:0023:0099f00c={KERNEL32
0168115b ba93f9a17f      mov     edx, 7FA1F993h
01681160 8b4df4          mov     ecx, dword ptr [ebp-0Ch]
01681163 e8e6f8ffff      call    01680a4e
01681168 89857cffffff    mov     dword ptr [ebp-84h], eax
0168116e ba6eefa37f      mov     edx, 7FA3EF6Eh
01681173 8b4df4          mov     ecx, dword ptr [ebp-0Ch]
01681176 e8d3f8ffff      call    01680a4e
0168117b 898578ffffff    mov     dword ptr [ebp-88h], eax
01681181 bafbf1e17f      mov     edx, 7FE1F1FBh
01681186 8b4df4          mov     ecx, dword ptr [ebp-0Ch]
01681189 e8c0f8ffff      call    01680a4e
```

Conclusion :- Dynamic address resolution of APIs is used by almost all evasive malware. It hinders the static analysis of malware and can be used in creative ways to even avoid API Logging.