

# AV engines evasion techniques - part 5. Simple C++ example.

[cocomelonc.github.io/tutorial/2022/03/22/simple-av-evasion-5.html](https://cocomelonc.github.io/tutorial/2022/03/22/simple-av-evasion-5.html)

March 22, 2022

4 minute read

Hello, cybersecurity enthusiasts and white hackers!

The screenshot shows a terminal session on a Kali Linux host (cocomelonc@kali) with a C++ file named 'hack.cpp' open. The code contains logic to find a specific API function by name and then call it. A message box is displayed on the Windows host (win7-x86) with the text 'Meow-meow!'.

```
File Actions Edit View Help
nc [cocomelonc@kali]:~...2-malware-av-evasion-5 x cocomelonc@kali:
29     PDWORD fAddr = (PDWORD)((LPBYTE)h + i
30     PDWORD fName = (PDWORD)((LPBYTE)h +
31     PWORD f0rd = (PWORD)((LPBYTE)h + img
32
33     for (DWORD i = 0; i < img_edt->Address
34         LPSTR pFuncName = (LPSTR)((LPBYTE)h
35
36     if (calcMyHash(pFuncName) == myHash
37         printf("successfully found! %s -
38         return (LPVOID)((LPBYTE)h + fAddr
39     }
40 }
41     return nullptr;
42 }
43
44 int main() {
45     HMODULE mod = LoadLibrary("user32.dll"
46     LPVOID addr = getAPIAddr(mod, 1703669
47     printf('0x%p\n', addr);
48     fnMessageBoxA myMessageBoxA = (fnMess
49     myMessageBoxA(NULL, "Meow-meow!", "-^
50     return 0;
51 }
```

win7-x86 (malware-test) [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Windows PowerShell

PS Z:\2022-03-22-malware-av-evasion-5> dir

Directory: Z:\2022-03-22-malware-av-evasion-5

Mode	LastWriteTime	Length	Name
d---	3/24/2022 12:51 PM	102	.git
-	3/25/2022 3:09 PM	102	meow.cpp
-	3/25/2022 3:09 PM	1382	meow.exe
-	3/25/2022 4:54 PM	39846	hack.exe
-	3/24/2022 12:50 PM	169	hack.cpp
-	3/25/2022 4:11 PM	1600	hack.pyc

PS Z:\2022-03-22-malware-av-evasion-5> successfully found! MessageBoxA - 0x7737eaff

Meow-meow!

OK

4:36 PM 3/25/2022

NORMAL hack.cpp cpp utf-8[unix] 100% In :51/51

This post is a result of my own research into another AV evasion trick. An example how to bypass AV engines in simple C++ malware.

## hashing function names

This is a simple but efficient technique for hiding WinAPI calls. It is **calling functions by hash names** and it's simple and often used in the “wild”.

Let's look at an example and you'll understand that it's not so hard.

## standard calling

Let's look at an example:

```
#include <windows.h>
#include <stdio.h>

int main() {
    MessageBoxA(NULL, "Meow-meow!", "=^..^=", MB_OK);
    return 0;
}
```

## Compile:

```
i686-w64-mingw32-g++ meow.cpp -o meow.exe -mconsole -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -Wint-to-pointer-cast -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
```

```
(cocomelonc㉿kali)-[~/projects/hacking/cybersec_blog/2022-03-22-malware-av-evasion-5]$ i686-w64-mingw32-g++ meow.cpp -o meow.exe -mconsole -I/usr/share/mingw-w64/include/ -s sections -fdata-sections -Wno-write-strings -Wint-to-pointer-cast -fno-exceptions -fmerge-s -static-libstdc++ -static-libgcc -fpermissive
In file included from /usr/share/mingw-w64/include/windows.h:70,
                 from meow.cpp:1:
/usr/share/mingw-w64/include/winbase.h:1066: warning: "InterlockedCompareExchangePointer"
1066 | #define InterlockedCompareExchangePointer __InlineInterlockedCompareExchangePointe
|
In file included 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 meow.cpp:1:
/usr/share/mingw-w64/include/winnt.h:2279: note: this is the location of the previous defi
2279 | #define InterlockedCompareExchangePointer(Destination, ExChange, Comperand) (PVOID)
InterlockedCompareExchange ((LONG volatile *) (Destination),(LONG) (LONG_PTR) (ExChange),(PTR) (Comperand))
|
(cocomelonc㉿kali)-[~/projects/hacking/cybersec_blog/2022-03-22-malware-av-evasion-5]$ ls -lht
total 68K
-rwxr-xr-x 1 cocomelonc cocomelonc 14K Mar 25 15:09 meow.exe
-rw-r--r-- 1 cocomelonc cocomelonc 102 Mar 25 15:09 meow.cpp
-rwxr-xr-x 1 cocomelonc cocomelonc 40K Mar 25 11:09 hack.exe
```

and run;

```
PS Z:\2022-03-22-malware-av-evasion-5> .\hack.exe
successfully found! MessageBoxA - 17036696
0x7237ea11
PS Z:\2022-03-22-malware-av-evasion-5> dir

Directory: Z:\2022-03-22-malware-av-evasion-5

Mode                LastWriteTime       Length Name
----                -----          ---- 
d-----        3/24/2022  12:51 PM           .git
-a----        3/25/2022  3:09 PM        102 meow.cpp
-a----        3/25/2022  3:09 PM      13824 meow.exe
-a----        3/24/2022  1:33 PM      40448 hack.exe
-a----        3/24/2022  12:50 PM       169 myhash.py
-a----        3/24/2022  1:33 PM      1699 hack.cpp

PS Z:\2022-03-22-malware-av-evasion-5> .\meow.exe
```

Meow-meow!!

OK

As expected, it's just a pop-up window.

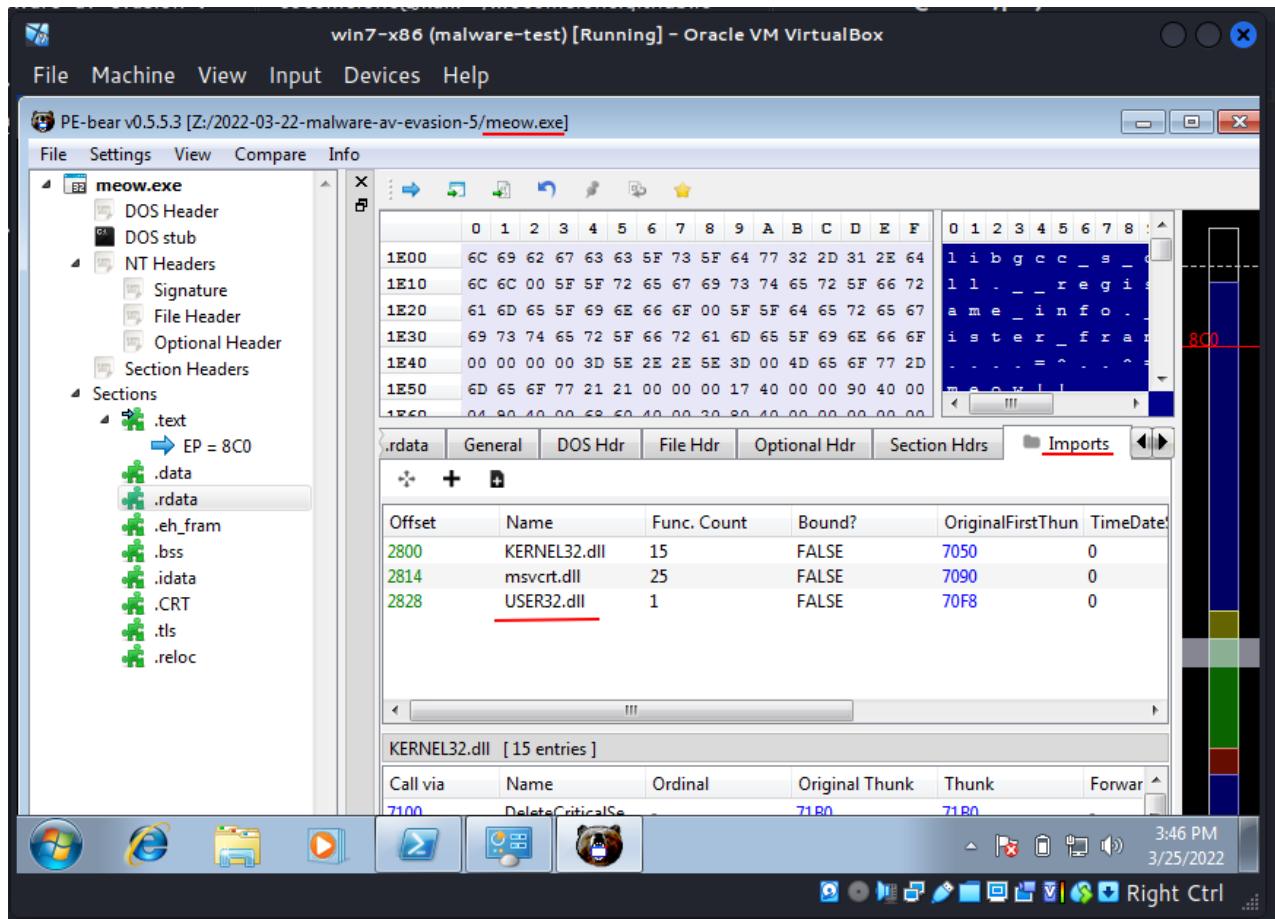
Then run `strings`:

```
strings -n 8 meow.exe | grep MessageBox
```

```
(cocomelonc㉿kali) - [~/projects/hacking/cyber
$ strings -n 8 meow.exe | grep MessageBox
MessageBoxA

(cocomelonc㉿kali) - [~/projects/hacking/cyber
$
```

As you can see, the WinAPI function are explicitly read in the basic static analysis and:



visible in the application's import table.

## hashing

Now let's hide the WinAPI function `MessageBoxA` we are using from malware analysts. Let's hash it:

```
# simple stupid hashing example
def myHash(data):
    hash = 0x35
    for i in range(0, len(data)):
        hash += ord(data[i]) + (hash << 1)
    print (hash)
    return hash

myHash("MessageBoxA")
```

and run it:

```
python3 myhash.py
```

```
(cocomelon㉿kali)-[~]$ python3 myhash.py
17036696
```

## practical example

---

What's the main idea? The main idea is we create code where we find WinAPI function address by it's hashing name via enumeration exported WinAPI functions.

First of all, let's declare a hash function identical in logic to the python code:

```
DWORD calcMyHash(char* data) {
    DWORD hash = 0x35;
    for (int i = 0; i < strlen(data); i++) {
        hash += data[i] + (hash << 1);
    }
    return hash;
}
```

Then, I declared function which find Windows API function address by comparing it's hash:

```
static LPVOID getAPIAddr(HMODULE h, DWORD myHash) {
    PIMAGE_DOS_HEADER img_dos_header = (PIMAGE_DOS_HEADER)h;
    PIMAGE_NT_HEADERS img_nt_header = (PIMAGE_NT_HEADERS)((LPBYTE)h + img_dos_header->e_lfanew);
    PIMAGE_EXPORT_DIRECTORY img_edt = (PIMAGE_EXPORT_DIRECTORY)((LPBYTE)h + img_nt_header->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);
    PDWORD fAddr = (PDWORD)((LPBYTE)h + img_edt->AddressOfFunctions);
    PDWORD fName = (PDWORD)((LPBYTE)h + img_edt->AddressOfNames);
    PWORD fOrd = (PWORD)((LPBYTE)h + img_edt->AddressOfNameOrdinals);

    for (DWORD i = 0; i < img_edt->AddressOfFunctions; i++) {
        LPSTR pFuncName = (LPSTR)((LPBYTE)h + fName[i]);

        if (calcMyHash(pFuncName) == myHash) {
            printf("successfully found! %s - %d\n", pFuncName, myHash);
            return (LPVOID)((LPBYTE)h + fAddr[fOrd[i]]));
        }
    }
    return nullptr;
}
```

The logic here is really simple. first we go through the PE headers to the exported functions we need. In the loop, we will look at and compare the hash passed to our function with the hashes of the functions in the export table and, as soon as we find a match, exit the loop:

```
//...
for (DWORD i = 0; i < img_edt->AddressOfFunctions; i++) {
    LPSTR pFuncName = (LPSTR)((LPBYTE)h + fNames[i]);

    if (calcMyHash(pFuncName) == myHash) {
        printf("successfully found! %s - %d\n", pFuncName, myHash);
        return (LPVOID)((LPBYTE)h + fAddr[fOrd[i]]);
    }
}
//...
```

Then we declare prototype of our function:

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

and `main()`:

```
int main() {
    HMODULE mod = LoadLibrary("user32.dll");
    LPVOID addr = getAPIAddr(mod, 17036696);
    printf("0x%p\n", addr);
    fnMessageBoxA myMessageBoxA = (fnMessageBoxA)addr;
    myMessageBoxA(NULL, "Meow-meow!", "=^..^=", MB_OK);
    return 0;
}
```

```
mc [cocomelonc@kali]:~...2-malware-av-evasion-5 x | cocomelonc@kali:~/pr.../cocomelonc.github.io x | cocomelonc@kali

└─(cocomelonc㉿kali)-[~/projects/hacking/cybersec_blog/2022-03-22-malw
└─$ python3 myhash.py
17036696
```

The terminal window shows the command `python3 myhash.py` being run, which outputs the value `17036696`. Below the terminal is a code editor window titled "hack.cpp". The code is a C++ program that searches for a specific hash value (17036696) in the memory of a process (user32.dll). If found, it prints the address of the function. The code uses `GetProcAddress` to find the address of the function by its name.

```
hack.cpp - ~/projects/hacking/cybersec_blog/2022-03-22-malw
File Edit View Selection Find Packages Help
hack.cpp x
31     PWORD f0rd = (PWORD)((LPBYTE)h + img_edt->AddressOfNameOrdinals);
32
33     for (DWORD i = 0; i < img_edt->AddressOfFunctions; i++) {
34         LPSTR pFuncName = (LPSTR)((LPBYTE)h + fNames[i]);
35
36         if (calcMyHash(pFuncName) == myHash) {
37             printf("successfully found! %s - %d\n", pFuncName, myHash);
38             return (LPVOID)((LPBYTE)h + fAddr[f0rd[i]]);
39         }
40     }
41     return nullptr;
42 }
43
44 int main() {
45     HMODULE mod = LoadLibrary("user32.dll");
46     LPVOID addr = getAPIAddr(mod, 17036696);
47     printf("0x%p\n", addr);
```

The full source code of our malware is:

```

/*
 * hack.cpp - hashing Win32API functions. C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/tutorial/2022/03/22/simple-malware-av-evasion-5.html
*/
#include <windows.h>
#include <stdio.h>

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

DWORD calcMyHash(char* data) {
    DWORD hash = 0x35;
    for (int i = 0; i < strlen(data); i++) {
        hash += data[i] + (hash << 1);
    }
    return hash;
}

static LPVOID getAPIAddr(HMODULE h, DWORD myHash) {
    PIMAGE_DOS_HEADER img_dos_header = (PIMAGE_DOS_HEADER)h;
    PIMAGE_NT_HEADERS img_nt_header = (PIMAGE_NT_HEADERS)((LPBYTE)h + img_dos_header->e_lfanew);
    PIMAGE_EXPORT_DIRECTORY img_edt = (PIMAGE_EXPORT_DIRECTORY)((LPBYTE)h + img_nt_header->OptionalHeader.DataDirectory[IMAGE_DIRECTORY_ENTRY_EXPORT].VirtualAddress);
    PDWORD fAddr = (PDWORD)((LPBYTE)h + img_edt->AddressOfFunctions);
    PDWORD fNames = (PDWORD)((LPBYTE)h + img_edt->AddressOfNames);
    PWORD  fOrd = (PWORD)((LPBYTE)h + img_edt->AddressOfNameOrdinals);

    for (DWORD i = 0; i < img_edt->AddressOfFunctions; i++) {
        LPSTR pFuncName = (LPSTR)((LPBYTE)h + fNames[i]);

        if (calcMyHash(pFuncName) == myHash) {
            printf("successfully found! %s - %d\n", pFuncName, myHash);
            return (LPVOID)((LPBYTE)h + fAddr[fOrd[i]]);
        }
    }
    return nullptr;
}

int main() {
    HMODULE mod = LoadLibrary("user32.dll");
    LPVOID addr = getAPIAddr(mod, 17036696);
    printf("0x%p\n", addr);
    fnMessageBoxA myMessageBoxA = (fnMessageBoxA)addr;
    myMessageBoxA(NULL, "Meow-meow!", "=^..^=", MB_OK);
    return 0;
}

```

```
}
```

## demo

Let's go to compile our malware `hack.cpp`:

```
i686-w64-mingw32-g++ hack.cpp -o hack.exe -mconsole -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -Wint-to-pointer-cast -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
```

```
└─(cocomelonc㉿kali)-[~/projects/hacking/cybersec_blog/2022-03-22-malware-av-evasion-5]
$ i686-w64-mingw32-g++ hack.cpp -o hack.exe -mconsole -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -Wint-to-pointer-cast -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
In file included from /usr/share/mingw-w64/include/windows.h:70,
                 from hack.cpp:6:
/usr/share/mingw-w64/include/winbase.h:1066: warning: "InterlockedCompareExchangePointer" redefined
1066 | #define InterlockedCompareExchangePointer __InlineInterlockedCompareExchangePointer
      |
In file included 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:6:
/usr/share/mingw-w64/include/winnt.h:2279: note: this is the location of the previous definition
2279 | #define InterlockedCompareExchangePointer(Destination, ExChange, Comperand) (VOID) (L
InterlockedCompareExchange ((LONG volatile *) (Destination),(LONG) (LONG_PTR) (ExChange),(LONG
PTR) (Comperand))
      |
      | MessageBoxA myMessageBoxA = (fnMessageBoxA)addr;
└─(cocomelonc㉿kali)-[~/projects/hacking/cybersec_blog/2022-03-22-malware-av-evasion-5]
$ ls -lt
total 48
-rwxr-xr-x 1 cocomelonc cocomelonc 40448 Mar 25 11:09 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc 1652 Mar 25 11:03 hack.cpp
```

and run:

```
.\hack.exe
```

The screenshot shows a terminal session on a Kali Linux host (cocomelonc㉿kali) running a C++ program named 'hack.cpp'. The program uses a debugger to find a specific memory address (0x737ea1) and then calls the Windows API function MessageBoxA with the message 'Meow-meow!' at that address. The output is displayed in a Windows 7 guest machine's PowerShell window, showing a message box with the text 'Meow-meow!' and an 'OK' button.

```

cocomelonc㉿kali:~/projects/hacking/cybersec_blog/2022-03-22-malware-av-evasion-5
File Actions Edit View Help
nc [cocomelonc㉿kali]: "...2-malware-av-evasion-5" cocomelonc㉿kali:
29 PDWORD fAddr = (PDWORD)((LPBYTE)h + i
30 PDWORD fName = (PDWORD)((LPBYTE)h +
31 PWORD fOrd = (PWORD)((LPBYTE)h + img
32
33 System for (DWORD i = 0; i < img_edt->Address
34     LPSTR pFuncName = (LPSTR)((LPBYTE)h +
35
36     if (calcMyHash(pFuncName) == myHash
37         printf("successfully found! %s -"
38         return (LPVOID)((LPBYTE)h + fAddr
39     }
40 }
41 return nullptr;
42 }
43
44 int main() {
45     HMODULE mod = LoadLibrary("user32.dll"
46     LPVOID addr = getAPIAddr(mod, 1703669
47     printf("0x%p\n", addr);
48     fnMessageBoxA myMessageBoxA = (fnMess
49     myMessageBoxA(NULL, "Meow-meow!", "="^
50     return 0;
51 }

NORMAL hack.cpp

```

This screenshot is identical to the one above, but with a red box highlighting the message box window in the Windows 7 guest machine's interface, emphasizing the successful execution of the exploit.

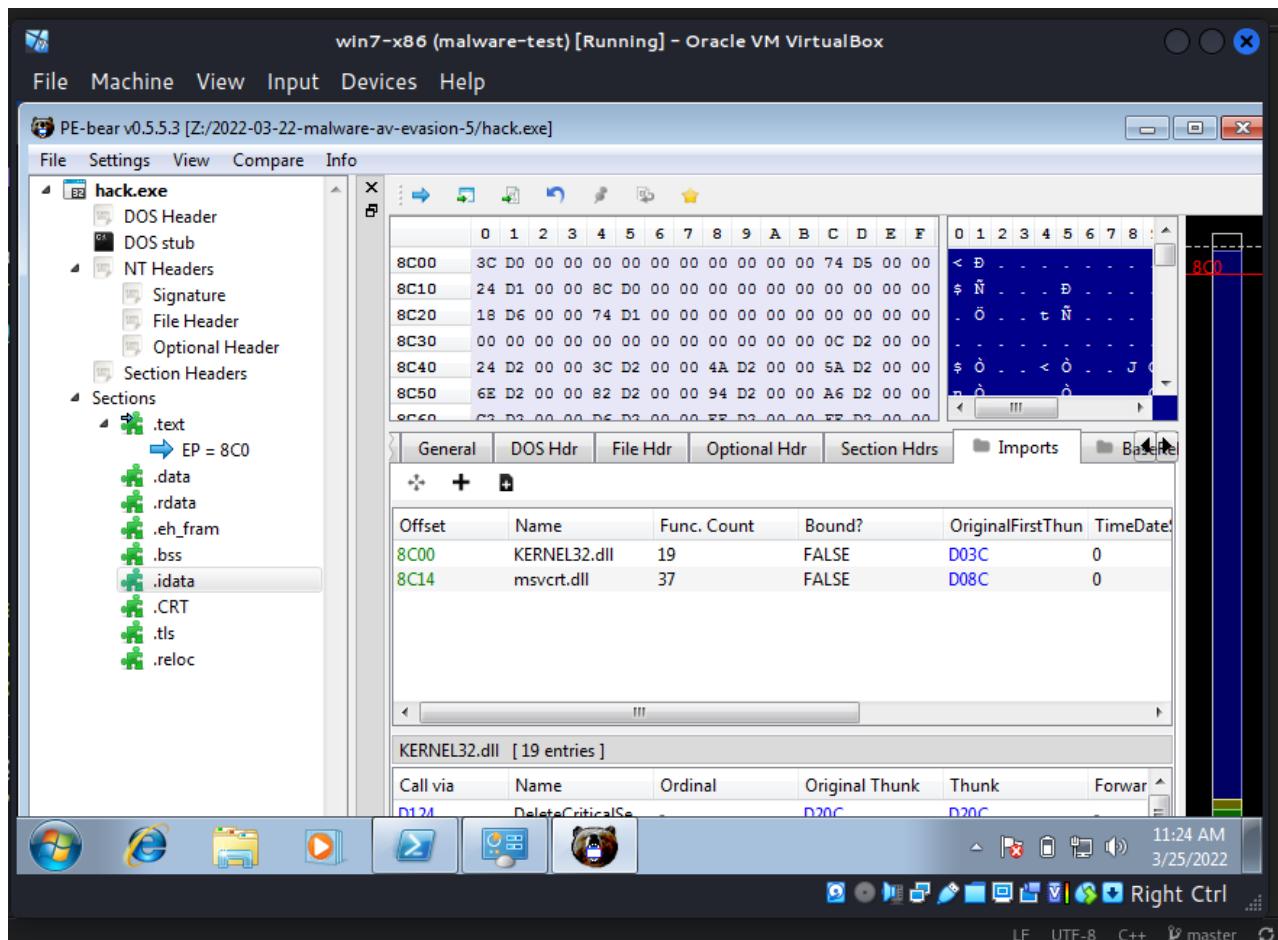
As you can see, our logic is worked!!! Perfect :)

What about **strings**?

```
strings -n 8 hack.exe | grep MessageBox
```

```
(py3)cocomelonc@Kali: /...log/cocomelonc.github.io ✘ cocomelonc@Kali: ~
32
└─(cocomelonc㉿kali)-[~/projects/hacking/cybersecurity]
$ hexdump -C hack.exe | grep "MessageBox"
35
└─(cocomelonc㉿kali)-[~/projects/hacking/cybersecurity]
$ strings -n 8 hack.exe | grep MessageBox
37
└─(cocomelonc㉿kali)-[~/projects/hacking/cybersecurity]
$ ┌─┐
39
└─┘
```

And let's go to see Import Address Table:



If we delve into the investigate of the malware, we, of course, will find our hashes, strings like `user32.dll`, and so on. But this is just a case study.

Let's go to upload to VirusTotal:

The screenshot shows the VirusTotal analysis interface for a file named 'd33210e3d7f9629d3465b2a0cec0c490d2254fa1b9a2fd047457bd9046bc0eee'. The file is identified as 'hack.exe' and has a 'peexe' extension. The file size is 39.00 KB and was submitted 1 minute ago at 2022-03-25 10:53:09 UTC. A summary indicates that 4 security vendors flagged the file as malicious. The detection table below shows the following results:

Engine	Detection	Details	Engine	Detection
Cylance	Unsafe		Cynet	Malicious (score: 100)
Ikarus	Trojan.Win32.Meterpreter		SecureAge APEX	Malicious
Acronis (Static ML)	Undetected		Ad-Aware	Undetected
AhnLab-V3	Undetected		Alibaba	Undetected
ALYac	Undetected		Anti-AVL	Undetected
Arcabit	Undetected		Avast	Undetected

<https://www.virustotal.com/gui/file/d33210e3d7f9629d3465b2a0cec0c490d2254fa1b9a2fd047457bd9046bc0eee/detection>

## So 4 of 65 AV engines detect our file as malicious

Notice that we evaded Windows Defender :)

But what about WinAPI functions in classic DLL injection?

I will self-research and write in a next post.

In real malware, hashes are additionally protected by mathematical functions and additionally encrypted.

| For example Carbanak uses several AV engines evasion techniques, one of them is WinAPI call hashing.

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

pe file format

Carbanak

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

