04 evasion: winapi hashing

Another AV evasion trick. An example how to bypass AV engines in simple C++ malware.

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

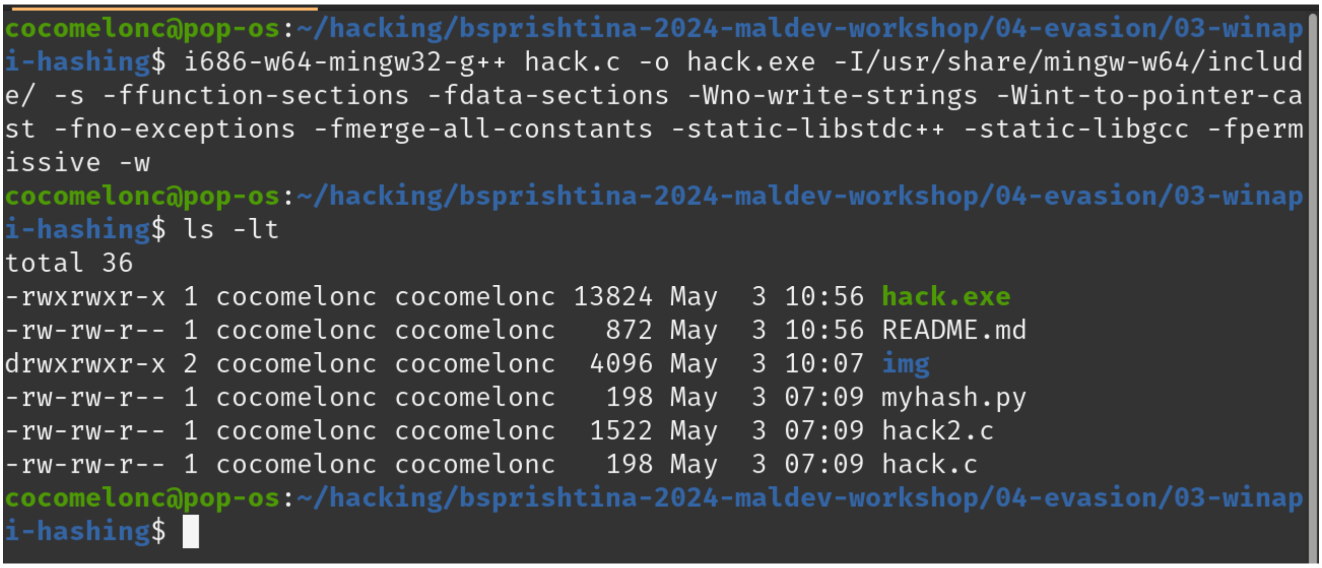
Let’s look at an example:

```
/*
 * malware AV evasion
 * hack.c - without hashing WINAPI functions.
 * author: @cocomelonc
*/
#include <windows.h>

int main() {
  MessageBoxA(NULL, "Hello, Prishtina!", "=^..^=", MB_OK);
  return 0;
}
```

Compile it:

```
i686-w64-mingw32-g++ hack.c -o hack.exe -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
```

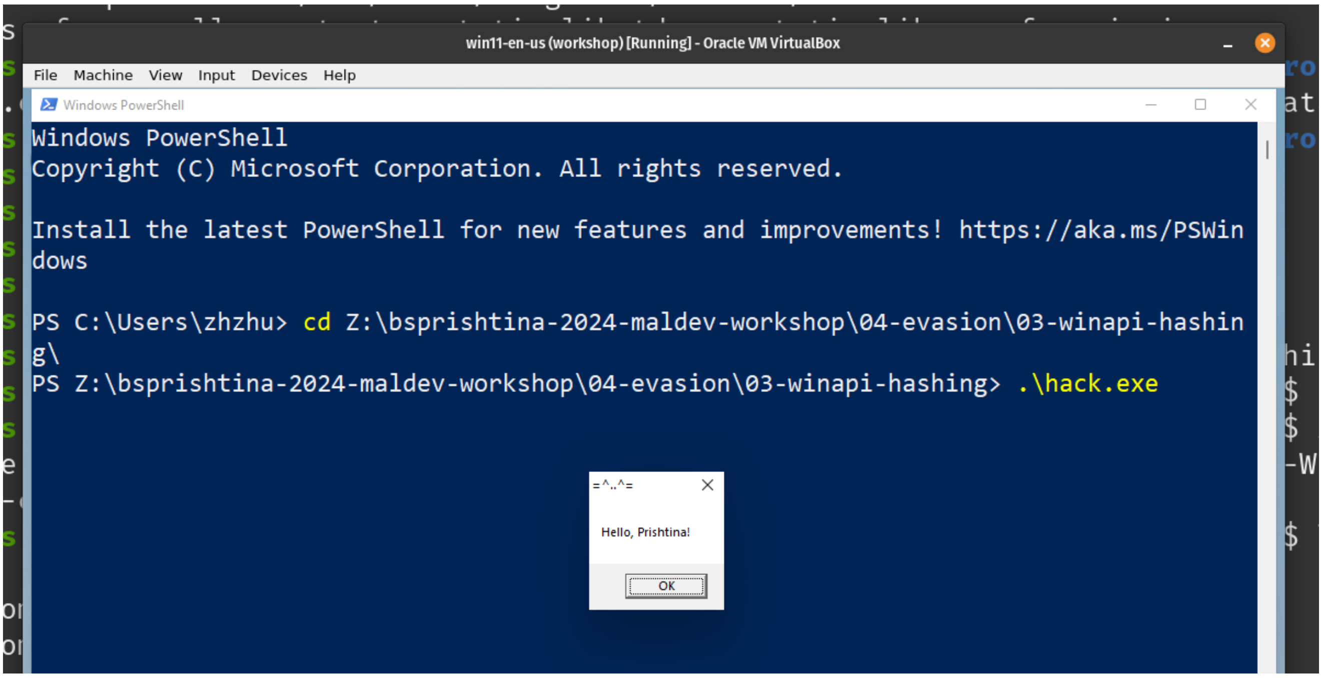
PROF



```
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/03-winapi-hashing$ i686-w64-mingw32-g++ hack.c -o hack.exe -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 -w
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/03-winapi-hashing$ ls -lt
total 36
-rwxrwxr-x 1 cocomelonc cocomelonc 13824 May  3 10:56 hack.exe
-rw-rw-r-- 1 cocomelonc cocomelonc   872 May  3 10:56 README.md
drwxrwxr-x 2 cocomelonc cocomelonc  4096 May  3 10:07 img
-rw-rw-r-- 1 cocomelonc cocomelonc   198 May  3 07:09 myhash.py
-rw-rw-r-- 1 cocomelonc cocomelonc  1522 May  3 07:09 hack2.c
-rw-rw-r-- 1 cocomelonc cocomelonc   198 May  3 07:09 hack.c
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/03-winapi-hashing$ 
```

Run it:

```
.\hack.exe
```


win11-en-us (workshop) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWin
dows

PS C:\Users\zhzhu> cd Z:\bsprishtina-2024-maldev-workshop\04-evasion\03-winapi-hashin
g\
PS Z:\bsprishtina-2024-maldev-workshop\04-evasion\03-winapi-hashing> .\hack.exe
```

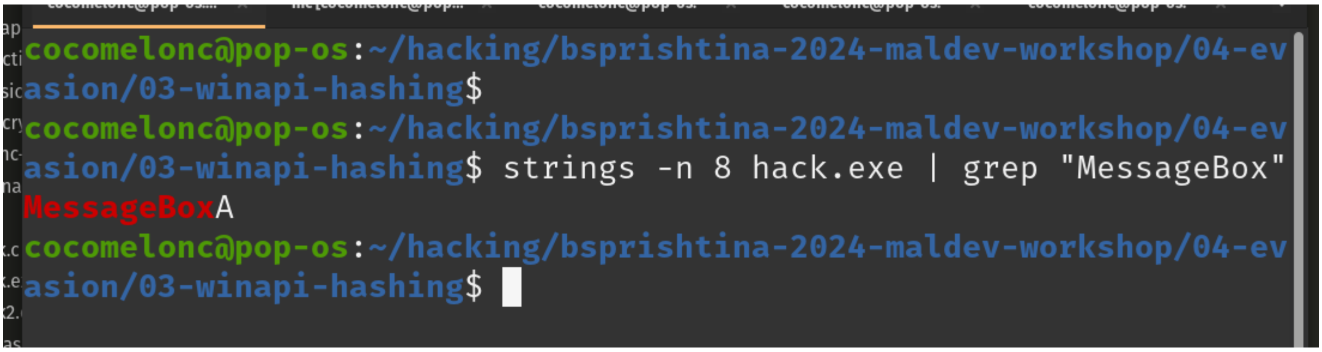
=^..^=

Hello, Prishtina!

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

Then run strings:

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

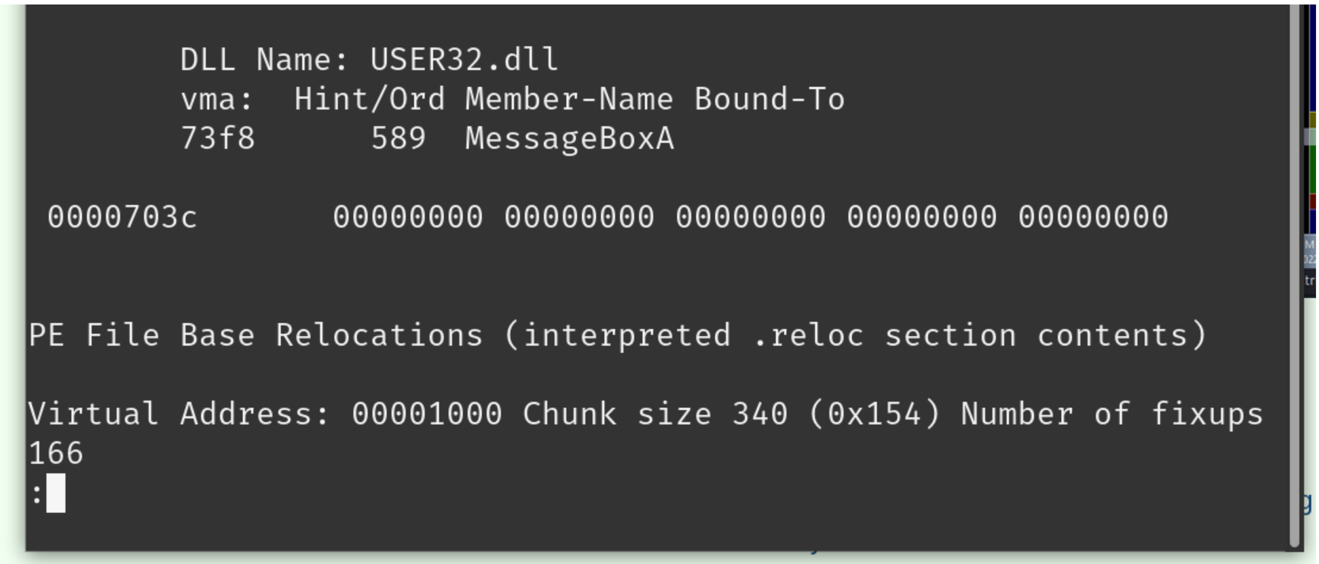


```
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-ev
asion/03-winapi-hashing$
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-ev
asion/03-winapi-hashing$ strings -n 8 hack.exe | grep "MessageBox"
MessageBoxA
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-ev
asion/03-winapi-hashing$
```

PROF

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

```
objdump -x -D hack.exe | less
```



```
        DLL Name: USER32.dll
        vma:  Hint/Ord Member-Name Bound-To
        73f8      589  MessageBoxA

 0000703c       00000000 00000000 00000000 00000000 00000000


PE File Base Relocations (interpreted .reloc section contents)

Virtual Address: 00001000 Chunk size 340 (0x154) Number of fixups
166
:
```

visible in the application's import table.

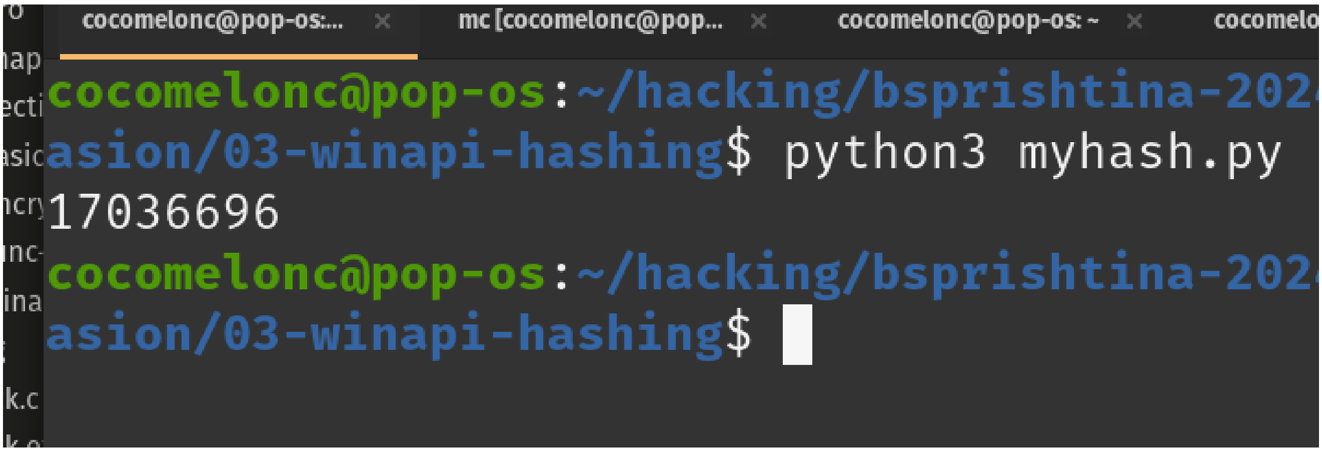
Now let's hide the WinAPI function `MessageBoxA` we are using from malware analysts. Let’s hash it:

```
# 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
```

PROF



```
cocomelonc@pop-os:~/hacking/bsprishtina-202
asion/03-winapi-hashing$ python3 myhash.py
17036696
cocomelonc@pop-os:~/hacking/bsprishtina-202
asion/03-winapi-hashing$ 
```

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].VirtualAddre
ss);
  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;
}
```

PROF

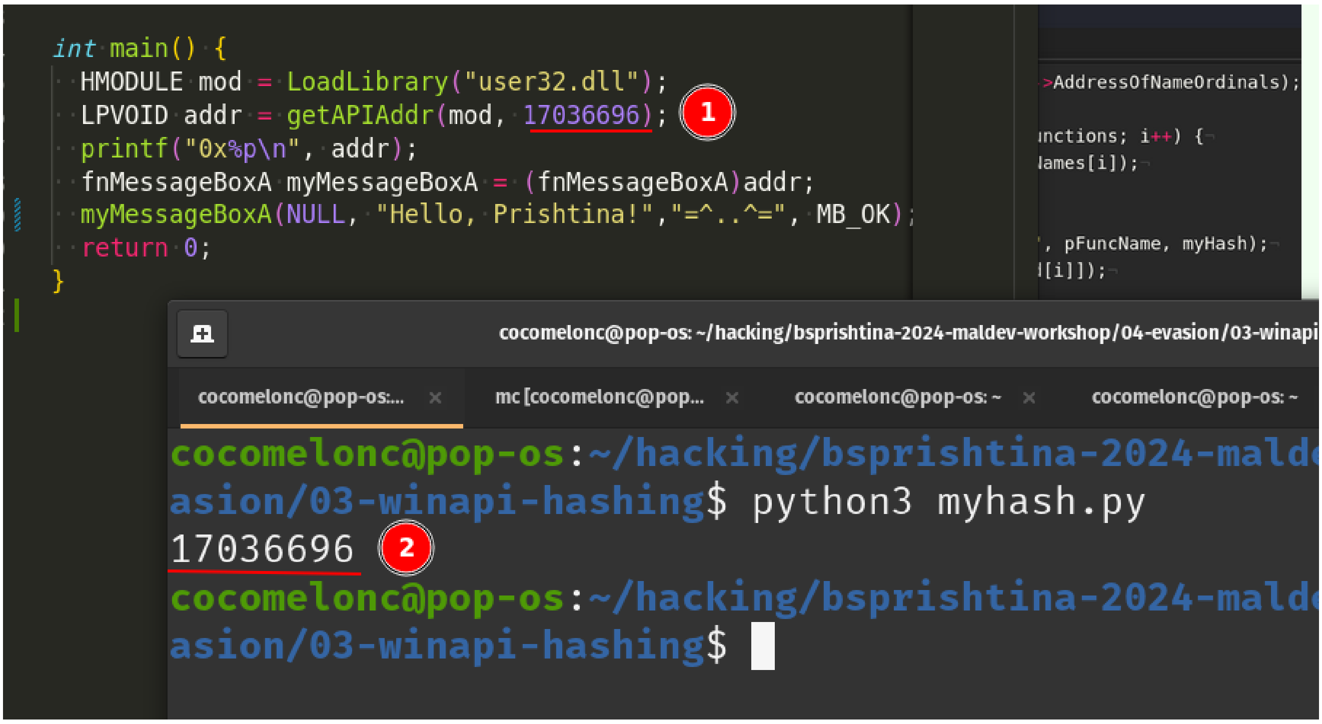
The logic here is really simple. first we go through the PE headers to the exported functions we need. In the loop, we will look at and compare the hash passed to our function with the hashes of the functions in the export table and, as soon as we find a match, exit the loop:

```
//...
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, "Hello, Prishtina!","=^..^=", MB_OK);
  return 0;
}
```

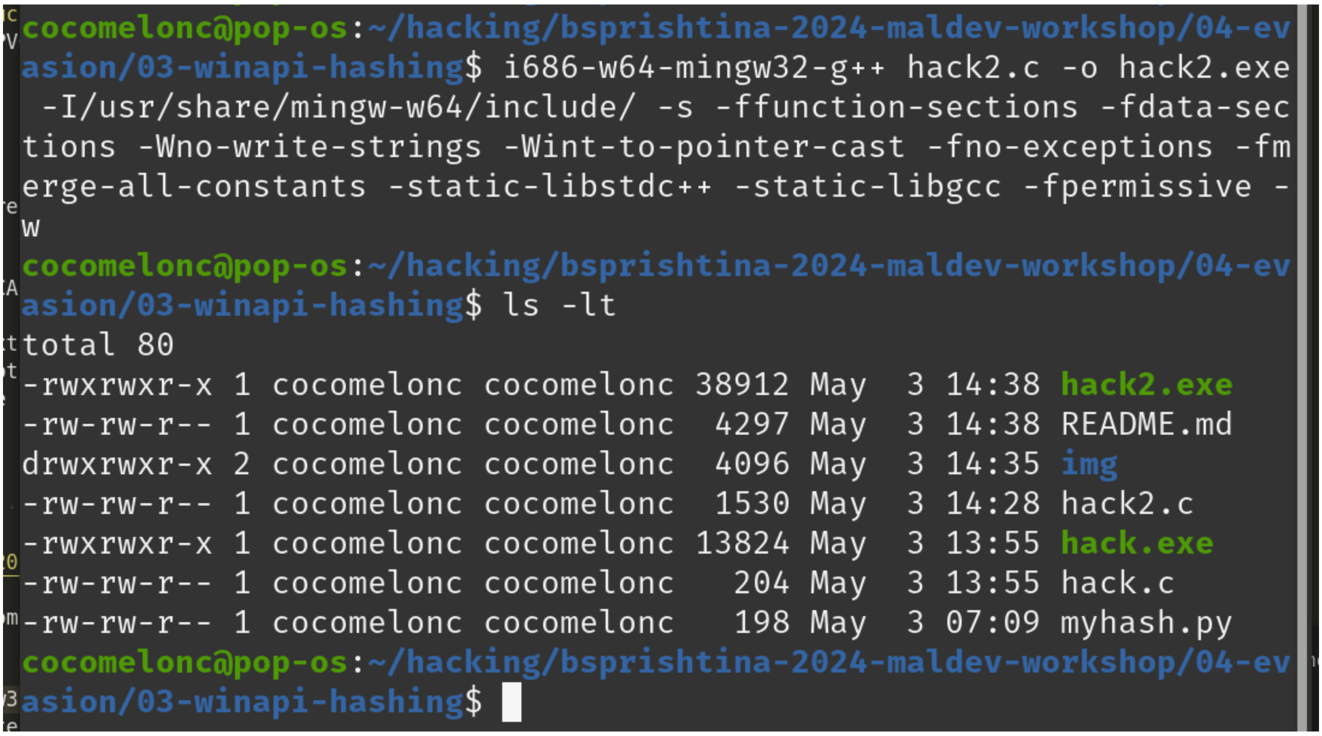
```
>AddressOfNameOrdinals);
unctions; i++) {
lames[i]);
', pFuncName, myHash);
l[i]]);
```

```
cocomelonc@pop-os:~/hacking/bsprishtina-2024-malde
asion/03-winapi-hashing$ python3 myhash.py
17036696
cocomelonc@pop-os:~/hacking/bsprishtina-2024-malde
asion/03-winapi-hashing$
```

PROF

Let's go to compile our modified malware:

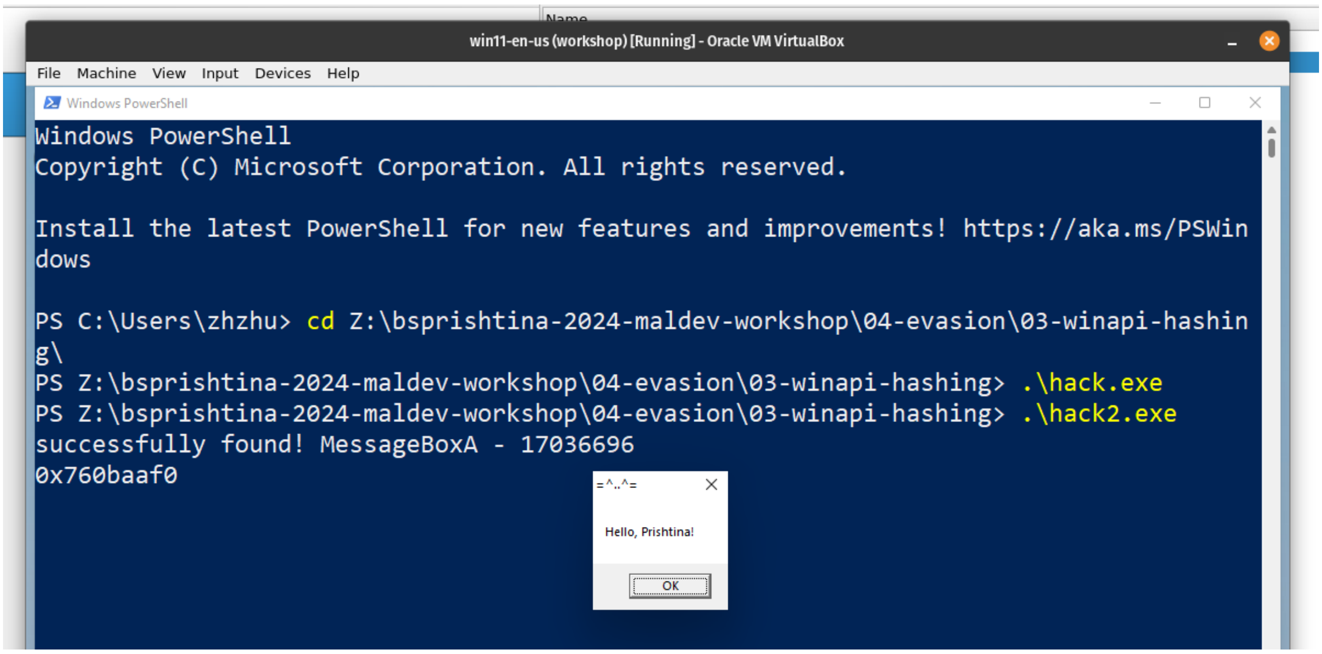
```
i686-w64-mingw32-g++ hack2.c -o hack2.exe -I/usr/share/mingw-
w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -
Wint-to-pointer-cast -fno-exceptions -fmerge-all-constants -static-
libstdc++ -static-libgcc -fpermissive
```



```
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/03-winapi-hashing$ i686-w64-mingw32-g++ hack2.c -o hack2.exe -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 -w
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/03-winapi-hashing$ ls -lt
total 80
-rwxrwxr-x 1 cocomelonc cocomelonc 38912 May  3 14:38 hack2.exe
-rw-rw-r-- 1 cocomelonc cocomelonc  4297 May  3 14:38 README.md
drwxrwxr-x 2 cocomelonc cocomelonc  4096 May  3 14:35 img
-rw-rw-r-- 1 cocomelonc cocomelonc  1530 May  3 14:28 hack2.c
-rwxrwxr-x 1 cocomelonc cocomelonc 13824 May  3 13:55 hack.exe
-rw-rw-r-- 1 cocomelonc cocomelonc   204 May  3 13:55 hack.c
-rw-rw-r-- 1 cocomelonc cocomelonc   198 May  3 07:09 myhash.py
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/03-winapi-hashing$ 
```

and run:

```
.\hack2.exe
```



win11-en-us (workshop) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\zhzhu> cd Z:\bsprishtina-2024-maldev-workshop\04-evasion\03-winapi-hashing\
PS Z:\bsprishtina-2024-maldev-workshop\04-evasion\03-winapi-hashing> .\hack.exe
PS Z:\bsprishtina-2024-maldev-workshop\04-evasion\03-winapi-hashing> .\hack2.exe
successfully found! MessageBoxA - 17036696
0x760baaf0
```

=^..^=

Hello, Prishtina!

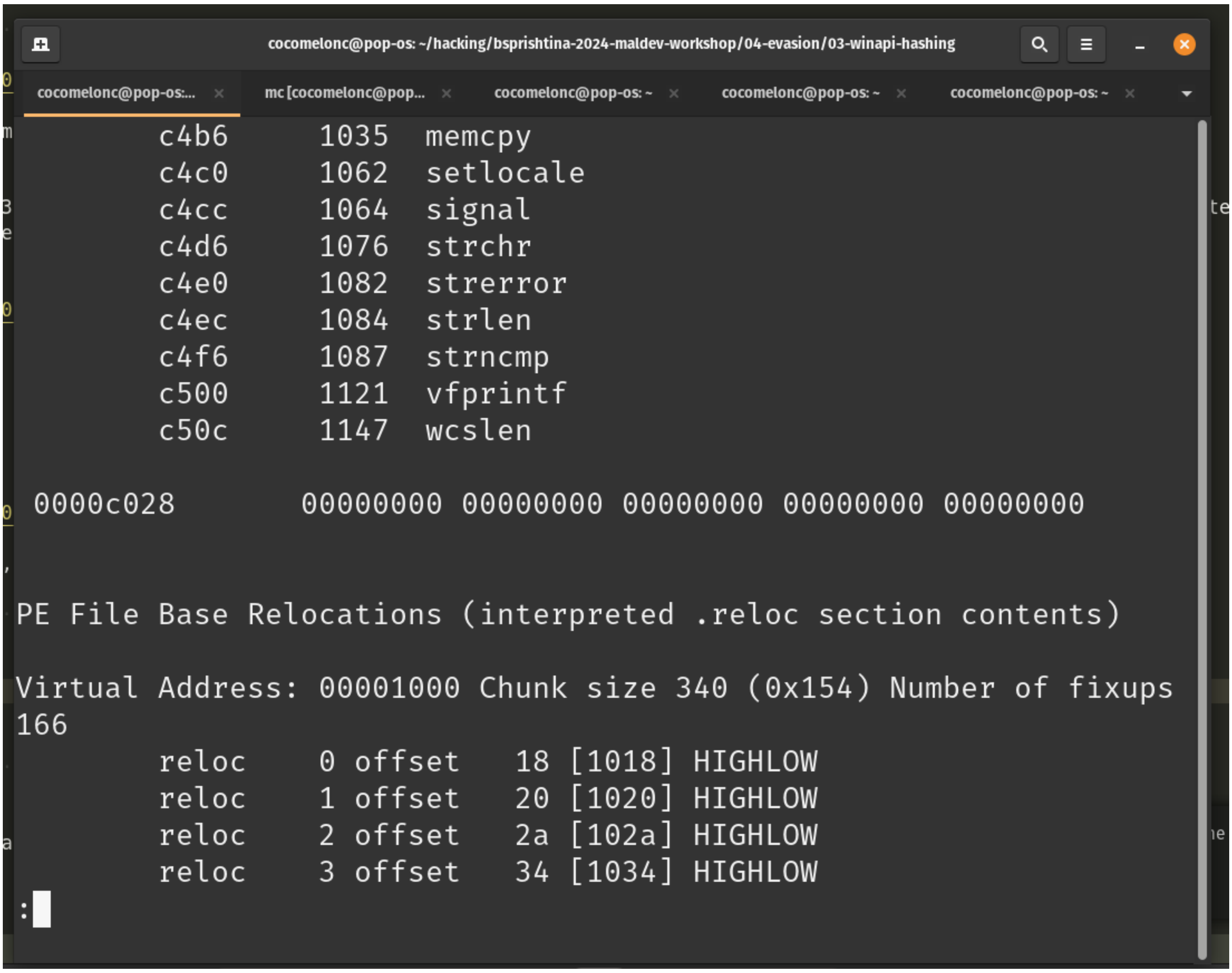
OK

PROF

As you can see, our logic is worked! Perfect 😃

Recheck IAT:

```
objdump -x -D hack2.exe | less
```

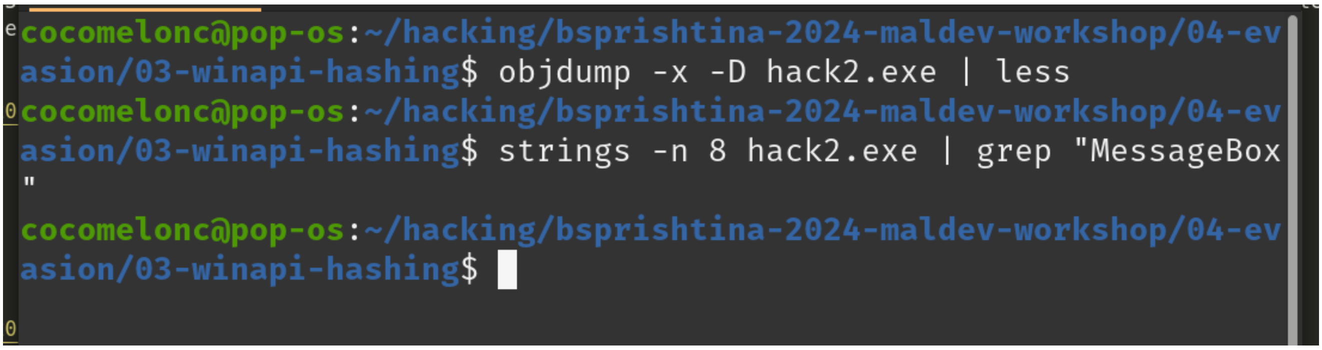


```
        c4b6      1035  memcpy
        c4c0      1062  setlocale
        c4cc      1064  signal
        c4d6      1076  strchr
        c4e0      1082  strerror
        c4ec      1084  strlen
        c4f6      1087  strncmp
        c500      1121  vfprintf
        c50c      1147  wcslen

 0000c028       00000000 00000000 00000000 00000000 00000000


PE File Base Relocations (interpreted .reloc section contents)

Virtual Address: 00001000 Chunk size 340 (0x154) Number of fixups
166
        reloc    0 offset   18 [1018] HIGHLOW
        reloc    1 offset   20 [1020] HIGHLOW
        reloc    2 offset   2a [102a] HIGHLOW
        reloc    3 offset   34 [1034] HIGHLOW
:
```

Recheck it via `strings`:

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

PROF



```
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/03-winapi-hashing$ objdump -x -D hack2.exe | less
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/03-winapi-hashing$ strings -n 8 hack2.exe | grep "MessageBox"
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/03-winapi-hashing$
```

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.

Many several years this trick is bypass Windows Defender's static analysis 😃