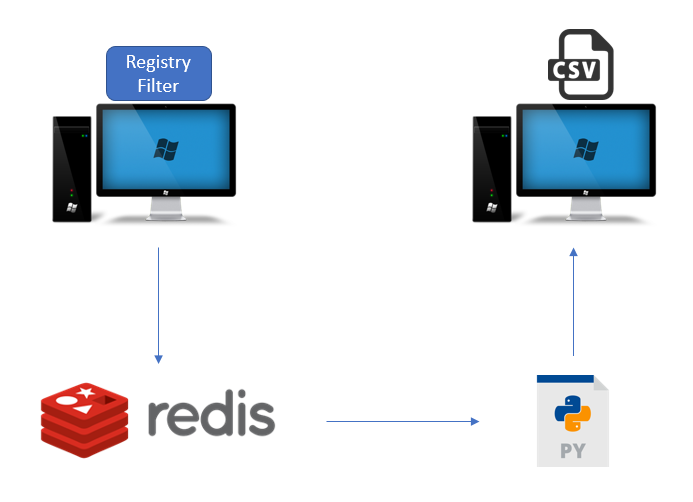
# High-Level Architecture



# Files in the Registry Filter Driver Project

**Under the exe folder:**

* regctrl.h
* common.h
* capture.c
* post.c
* pre.c
* regctrl.c
* util.c

**Under the sys folder:**

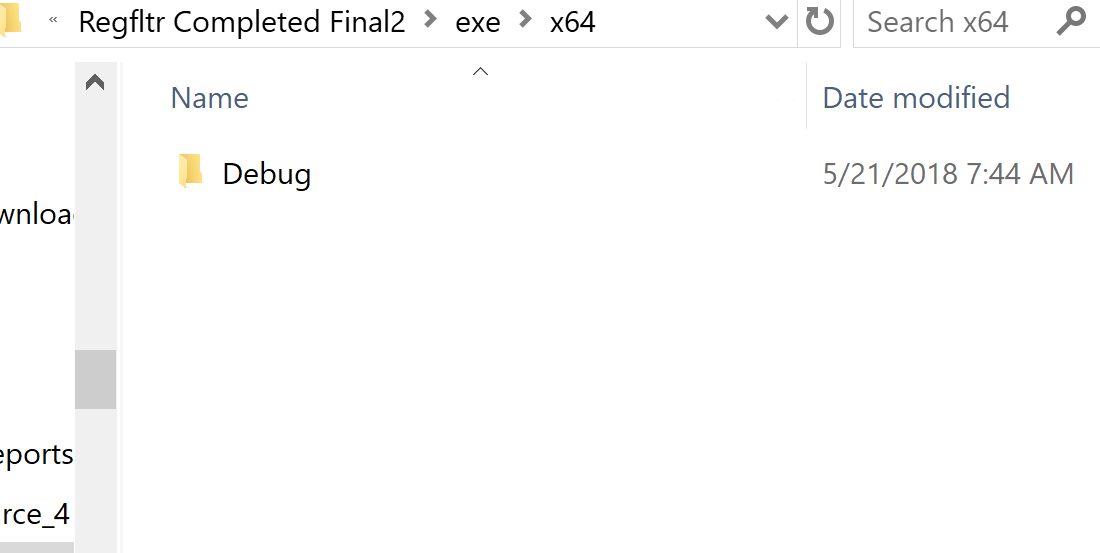
* tools.h
* regfltr.h
* log.h
* logformat.h
* Capture.c
* context.c
* driver.c
* Log.c
* MultiAlt.c
* Post.c
* Pre.c
* regfltr.c
* tool.c
* txr.c
* txrutil.c
* Util.c
* version.c

**Requirements:**

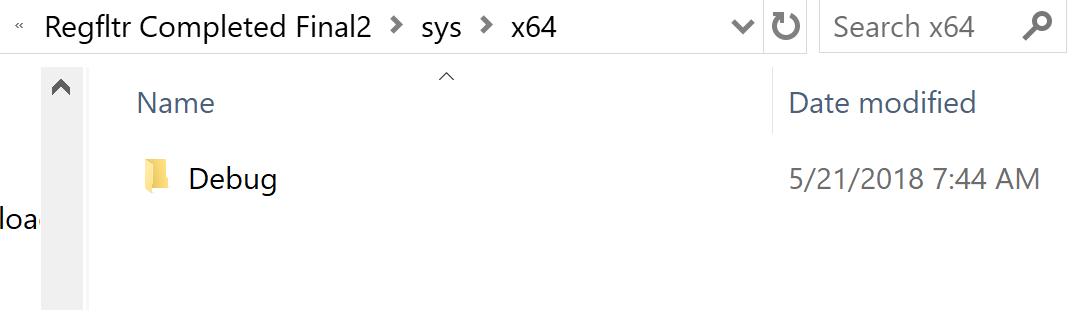
* Windows 10 SDK
* Windows 10 WDK
* DebugView

# Running the Windows Registry Filter Driver

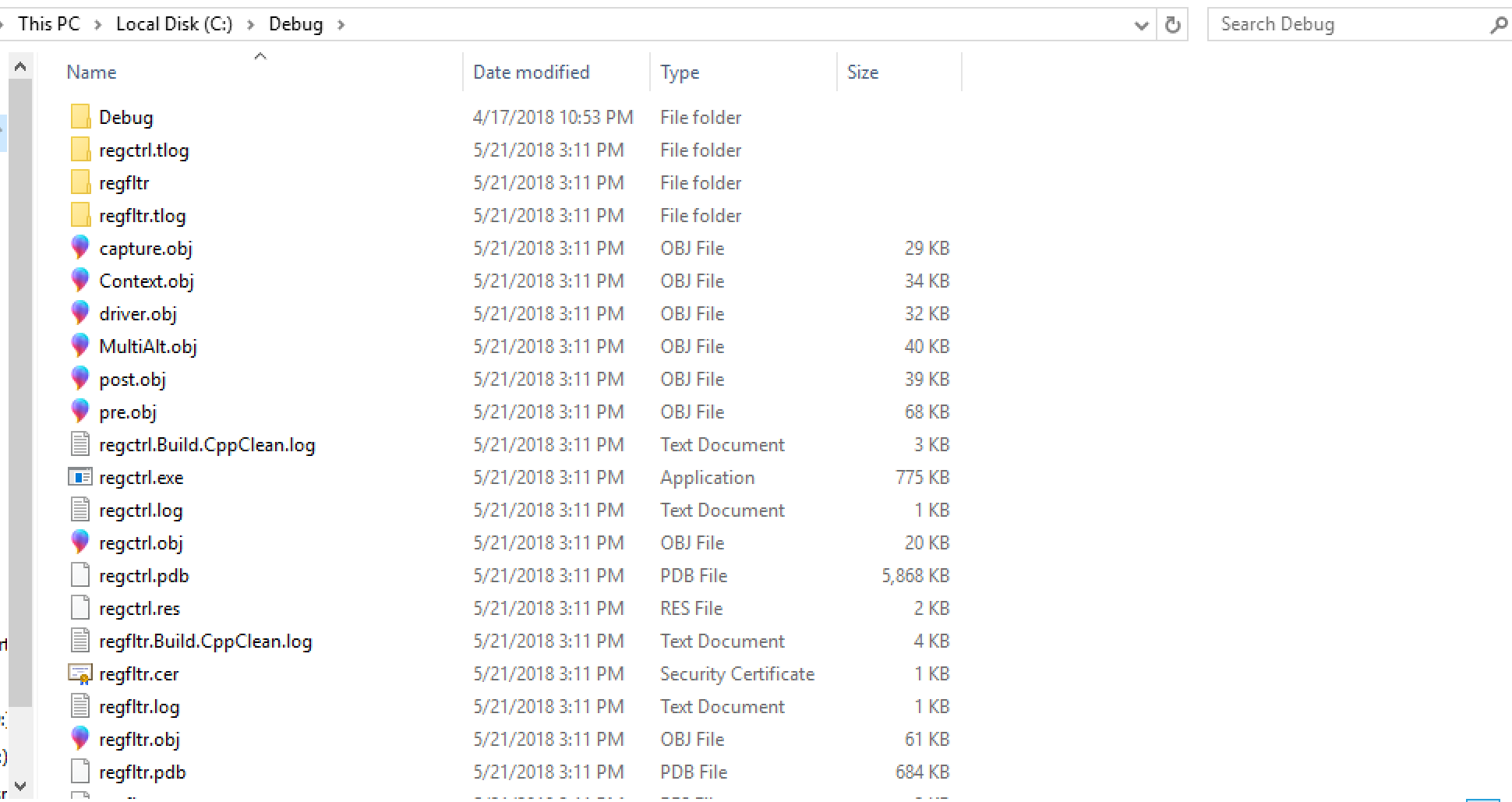
1. In the host machine, build the Visual Studio project files for “regctrl” and “regfltr” under the folders “Exe” and “Sys” respectively in the solution “regfltr”.
2. In the virtual machine used to test the filter driver, combine all the files and folders in “regfltr/exe/x64/Debug” and “regfltr/sys/x64/Debug” into one single folder.



(exe debug folder)

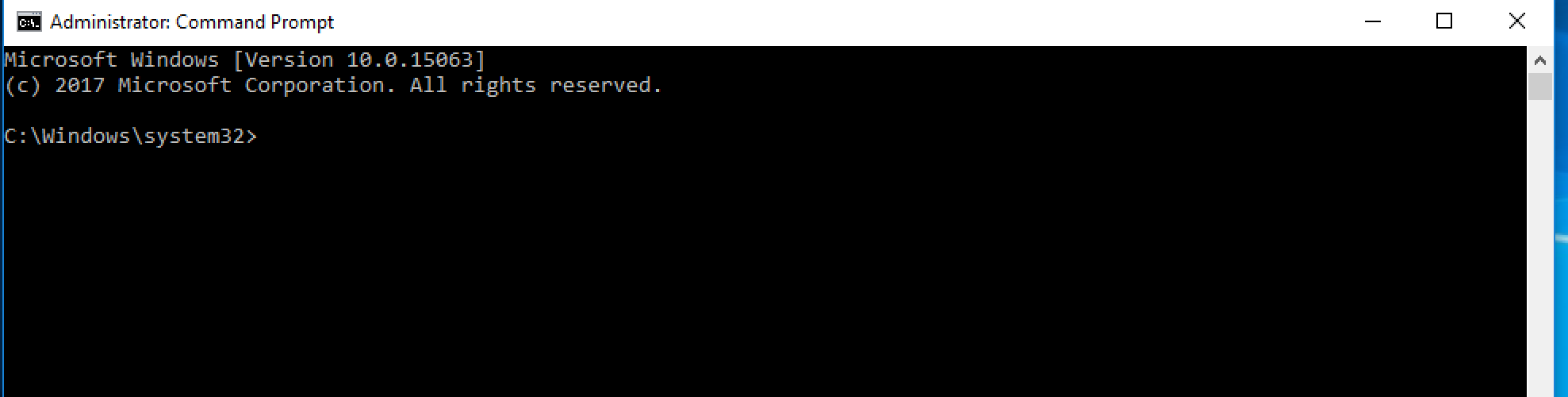


(sys debug folder)

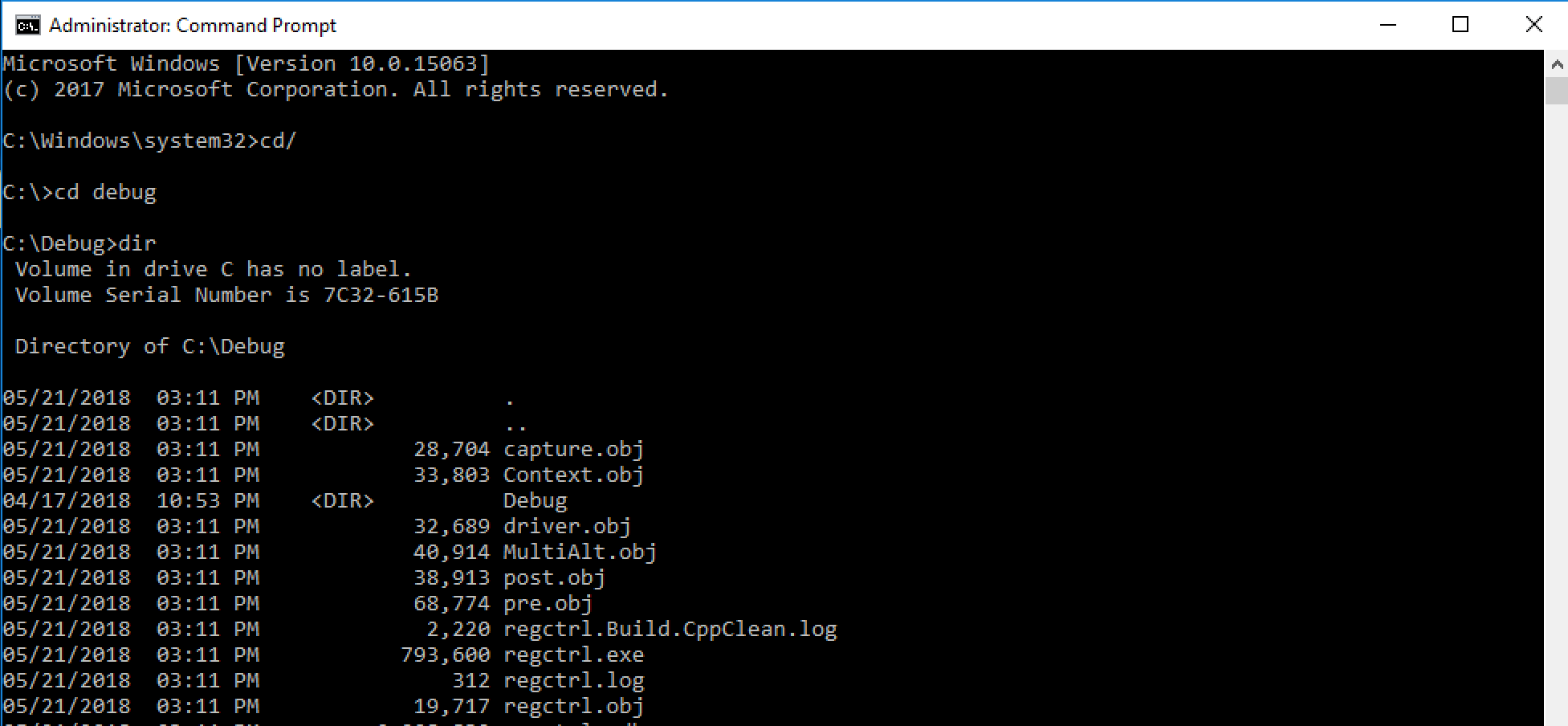


(exe and sys debug folder contents combined in single folder in testing machine)

1. In the virtual machine, open a command prompt window in administrator mode.



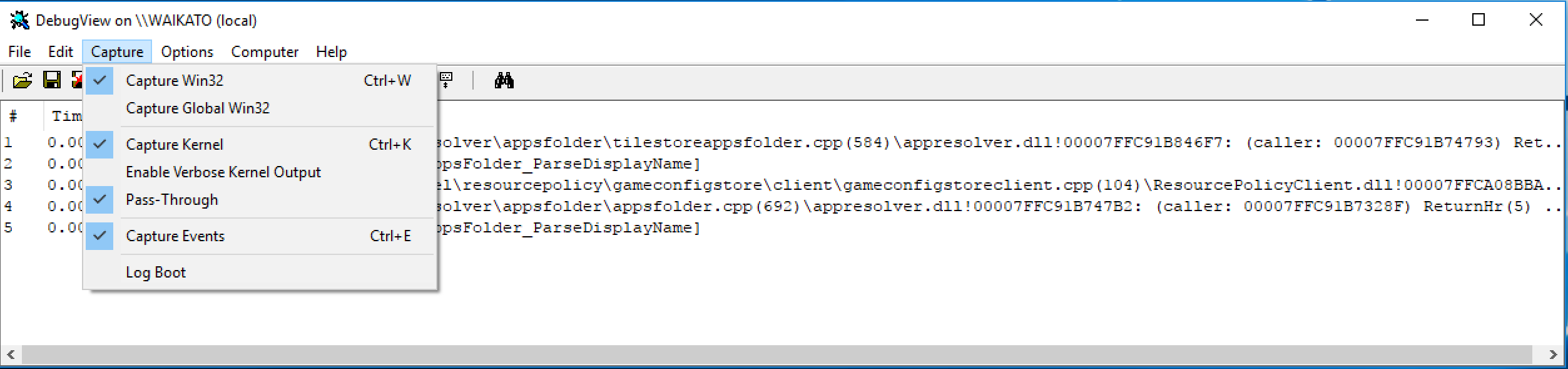
1. In the command prompt window, navigate to the folder where all the files and folders are stored in during step 2.



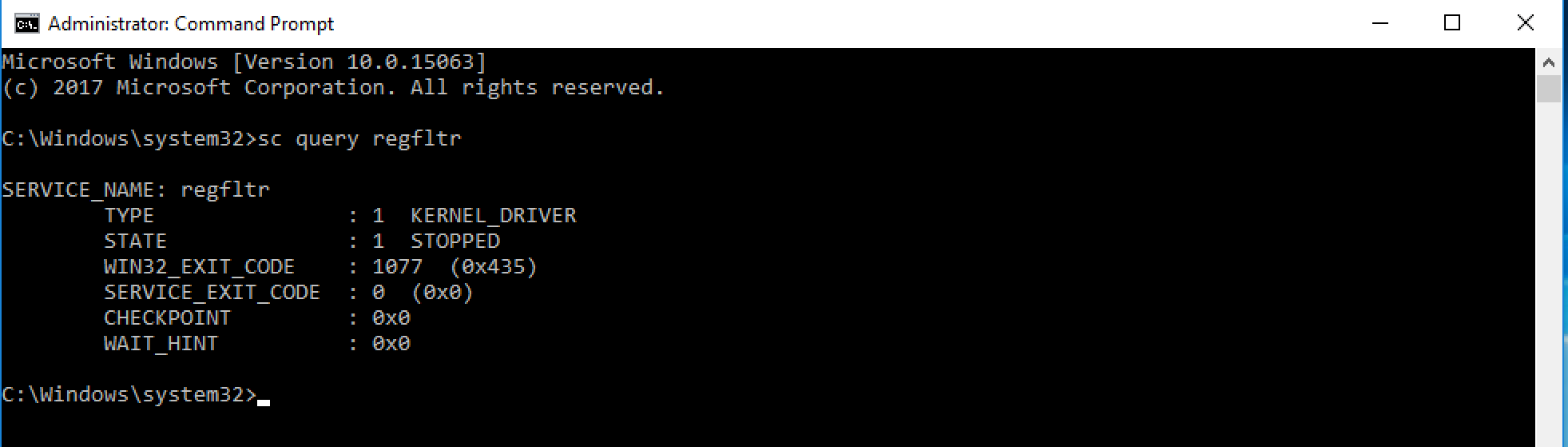
1. In the command prompt window, enter “regctrl.exe” to run the filter driver.



1. If you want to see the output of the filter driver, run the program “DebugView”, which should be placed in the virtual machine, in administrator mode. Under “Capture”, ensure that “Capture Win32”, “Capture Kernel”, “Enable Verbose Kernel Output”, “Pass-Through” and “Capture Events” are checked.



1. Once the filter driver has started running in the virtual machine, the virtual machine must be restarted to stop the filter driver before the files in the combined folder on the virtual machine can be replaced and the filter driver is run again. To check if the filter driver is running, perform an “sc query”.

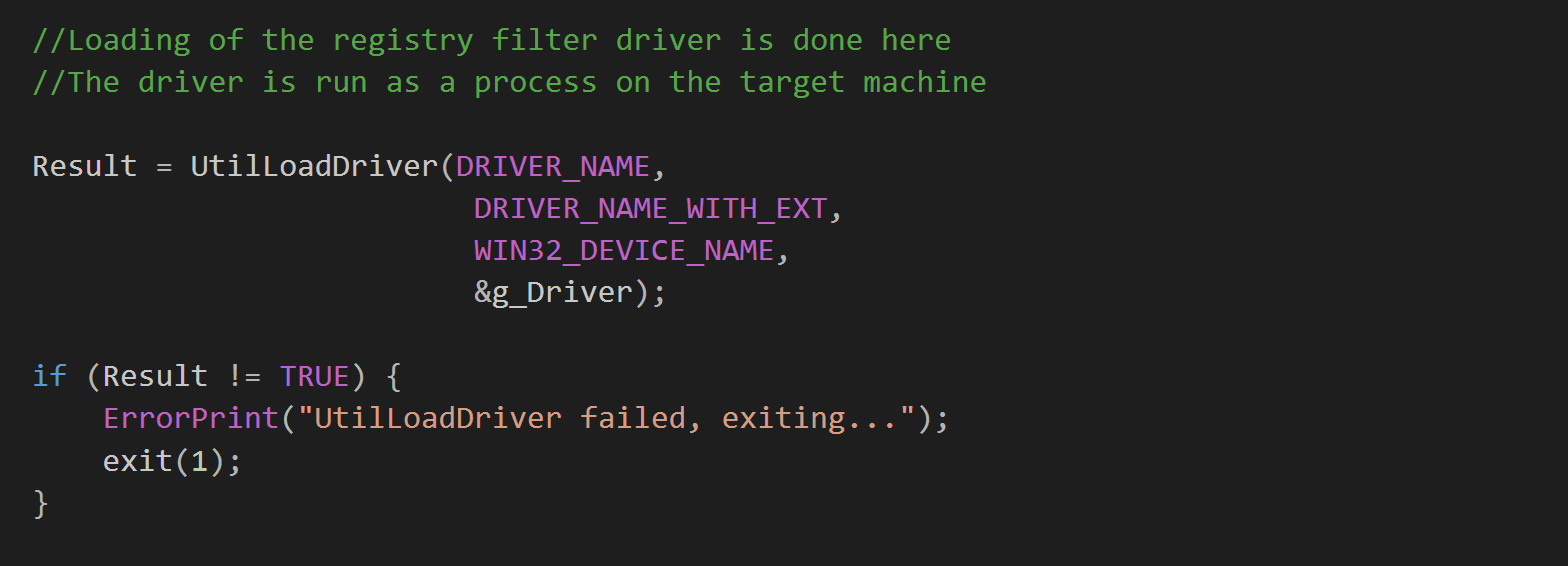


(Example of how to query the filter driver process)

# Important Parts of Code

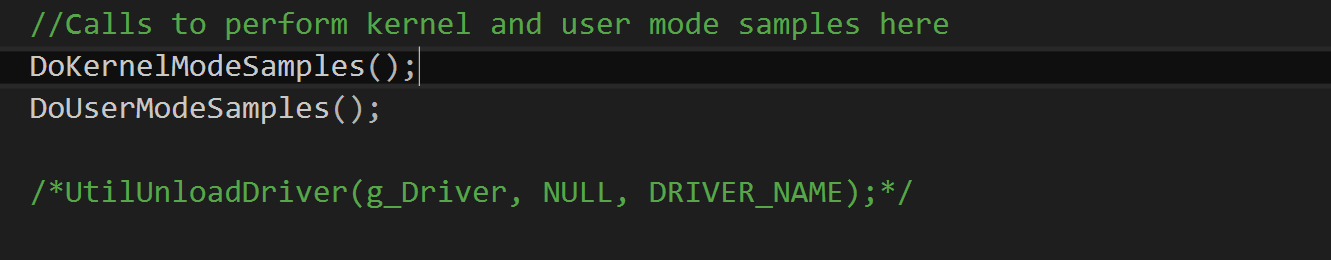
## Loading of Filter Driver

* File: regctrl.c (regfltr/Exe/regctrl.c)
* The function call to load the filter driver when the “regctrl.exe” is run is made in this file.
* The function called is “UtilLoadDriver()”



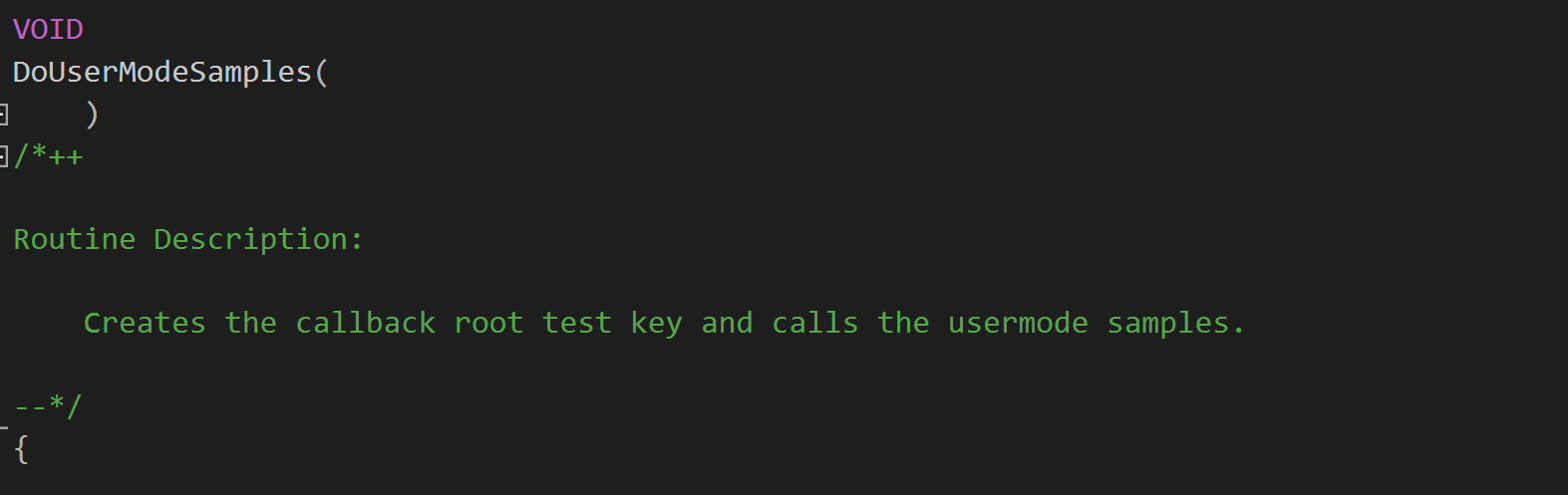
## Calling of Kernel and User Mode Samples

* File: regctrl.c (regfltr/Exe/regctrl.c)
* The calls to the functions “DoKernelModeSamples” and “DoUserModeSamples” are made in this file.

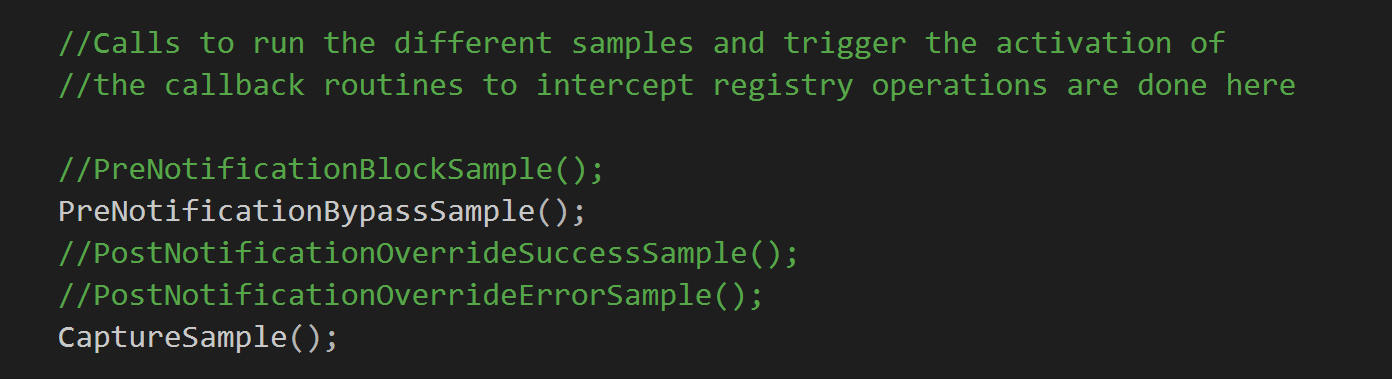


## Calling of Pre-Notification Bypass Samples

* File: regctrl.c (regfltr/Exe/regctrl.c)
* The call to run the pre-notification bypass sample, which registers the callback routine to intercept registry operations in the pre-notification phase, is made in this file in the function “DoUserModeSamples()”.

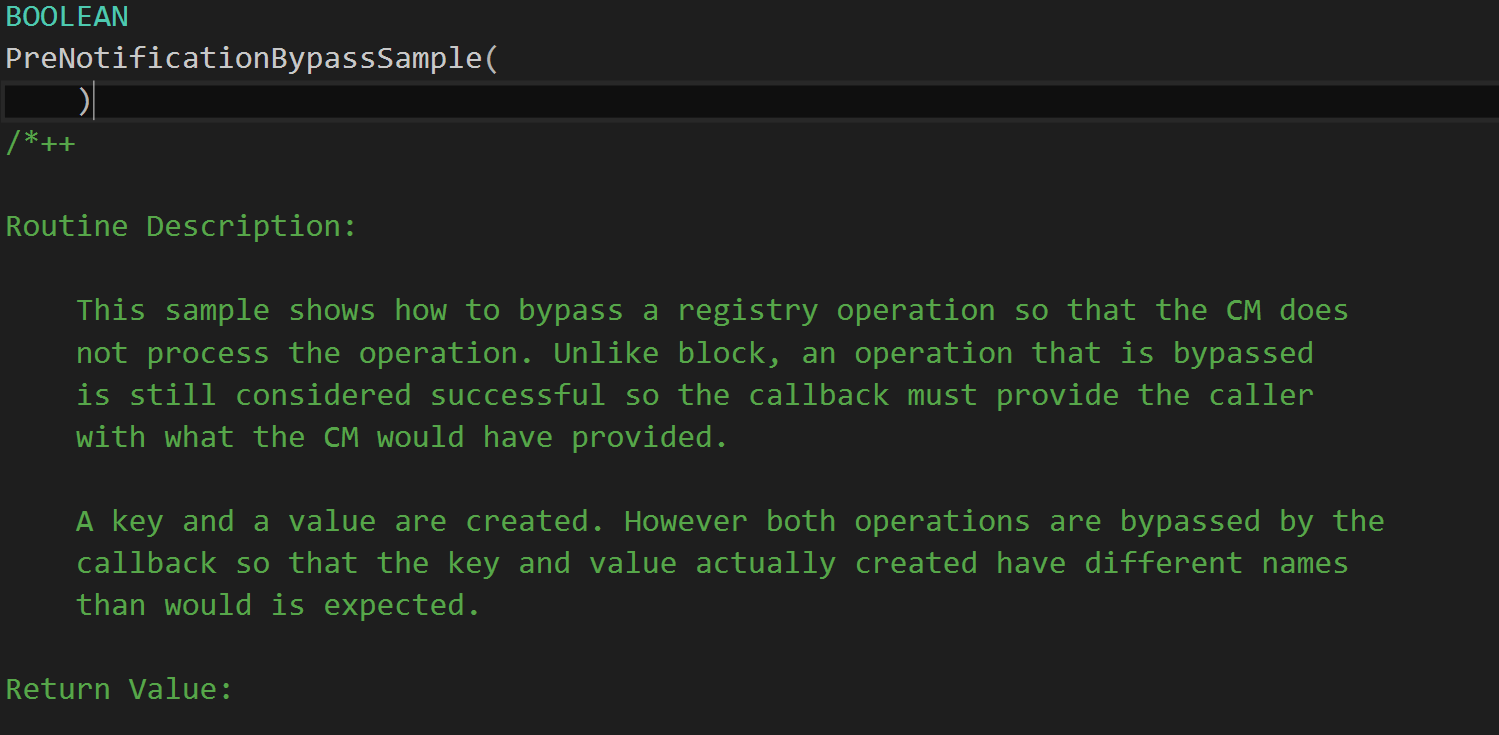


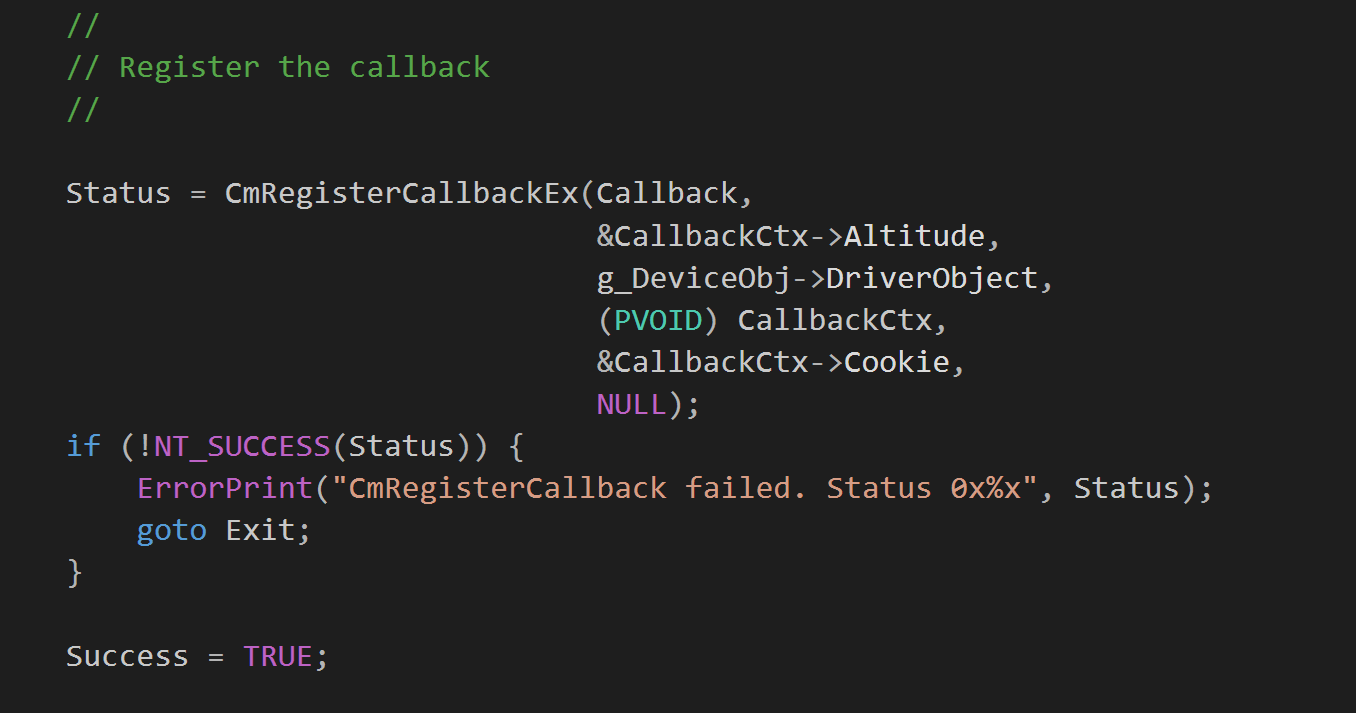
* The function called is “PreNotificationBypassSample()”.



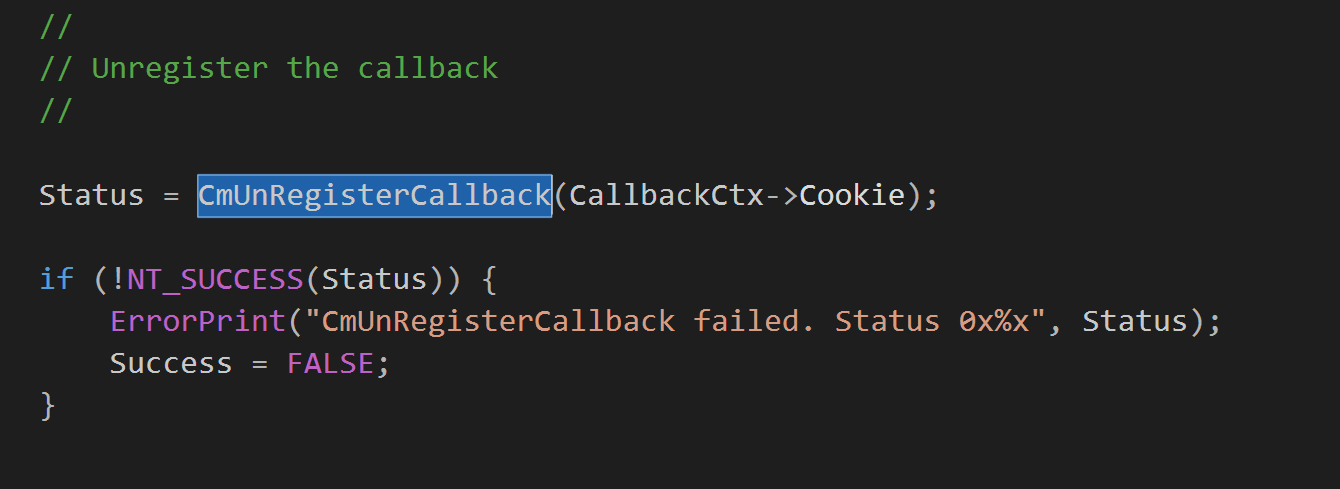
## Registering of Pre-Notification Callback Routine

* File: Pre.c (regfltr/Sys/Pre.c)
* The call to the function “CmRegisterCallbackEx” which registers the callback routine used to intercept registry operations is made in the function “PreNotificationBypassSample” in this file.



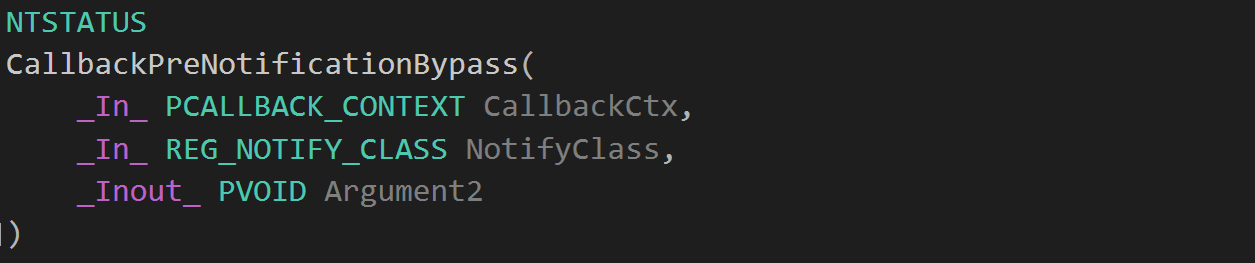


* The call to the function “CmUnRegisterCallback” which unregisters the callback routine is also done in the same function.

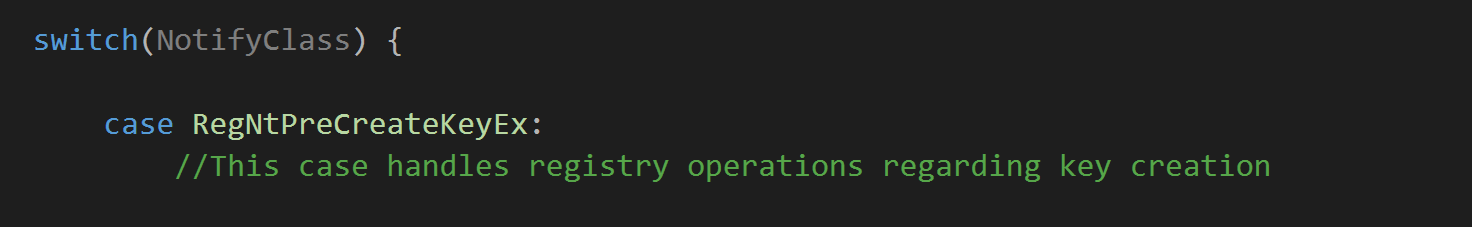


## Intercepting Registry Operations in Pre-Notification Phase

* File: Pre.c (regfltr/Sys/Pre.c)
* Pre-notification phase means that the registry operations have not actually taken effect in the registry yet.
* The code used to handle the extraction of information from the intercepted registry operations is made in the function “CallbackPreNotificationBypass”.

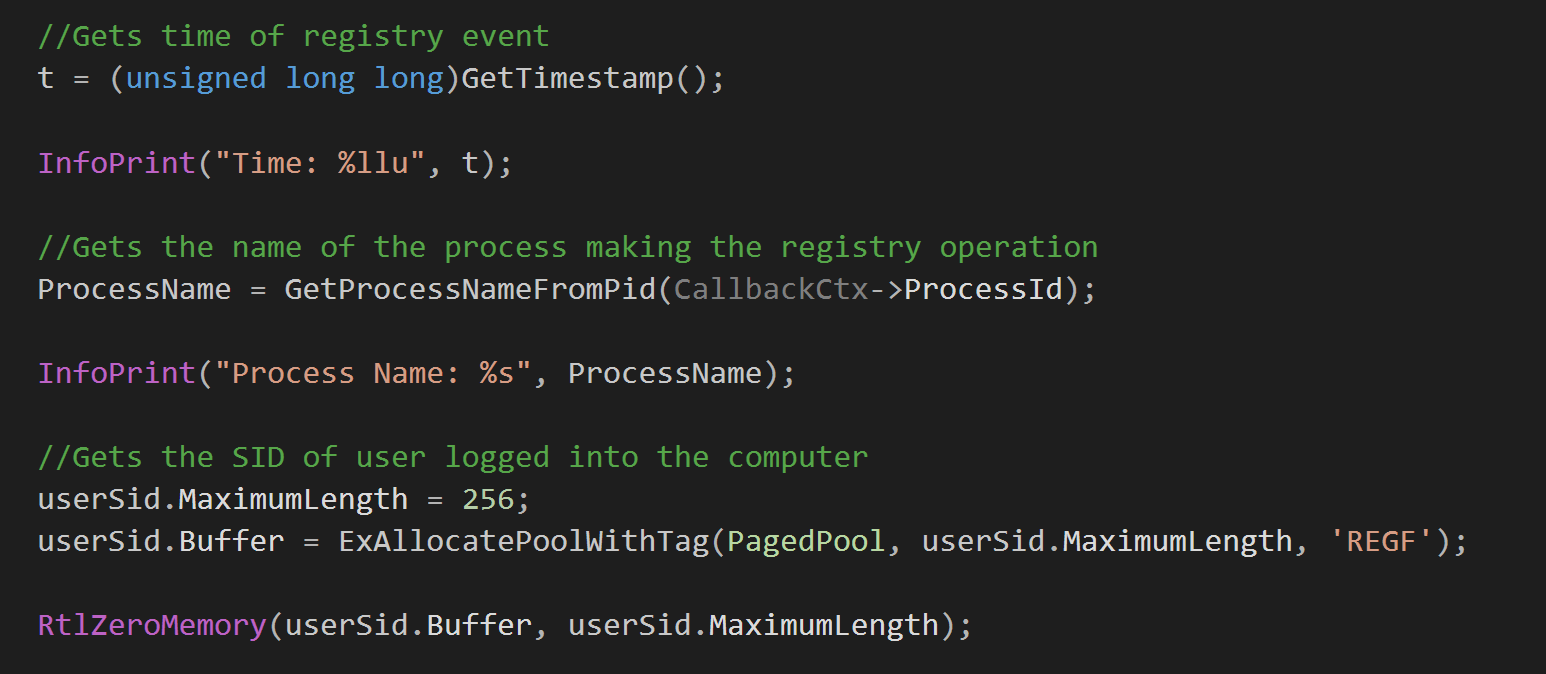


* The registry operation type checking is handled by a switch case, in which the cases are registry operation types under the REG\_NOTIFY\_CLASS enumeration type.

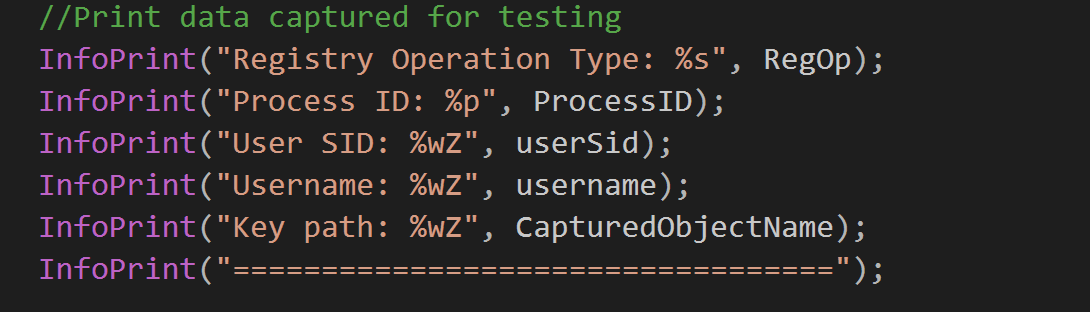


(Example of a case for a registry operation)

* The information extracted from the registry operations is stored in variables as declared in the function. Refer to comments in the code for more information about information collected and functions used to collect these information.

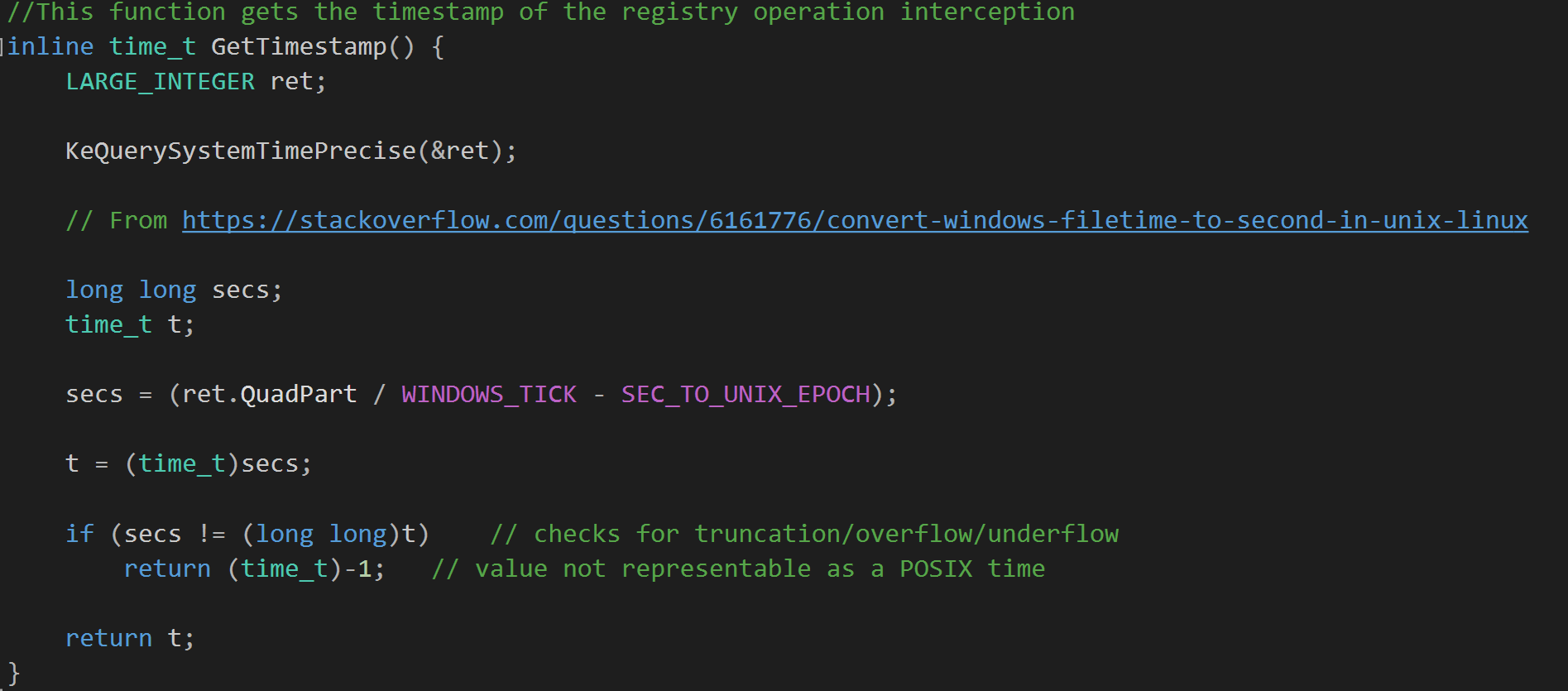


* The information collected for each operation is also printed out to the debugger using InfoPrint() for easy debugging and checking of the information collected about each registry operation intercepted.

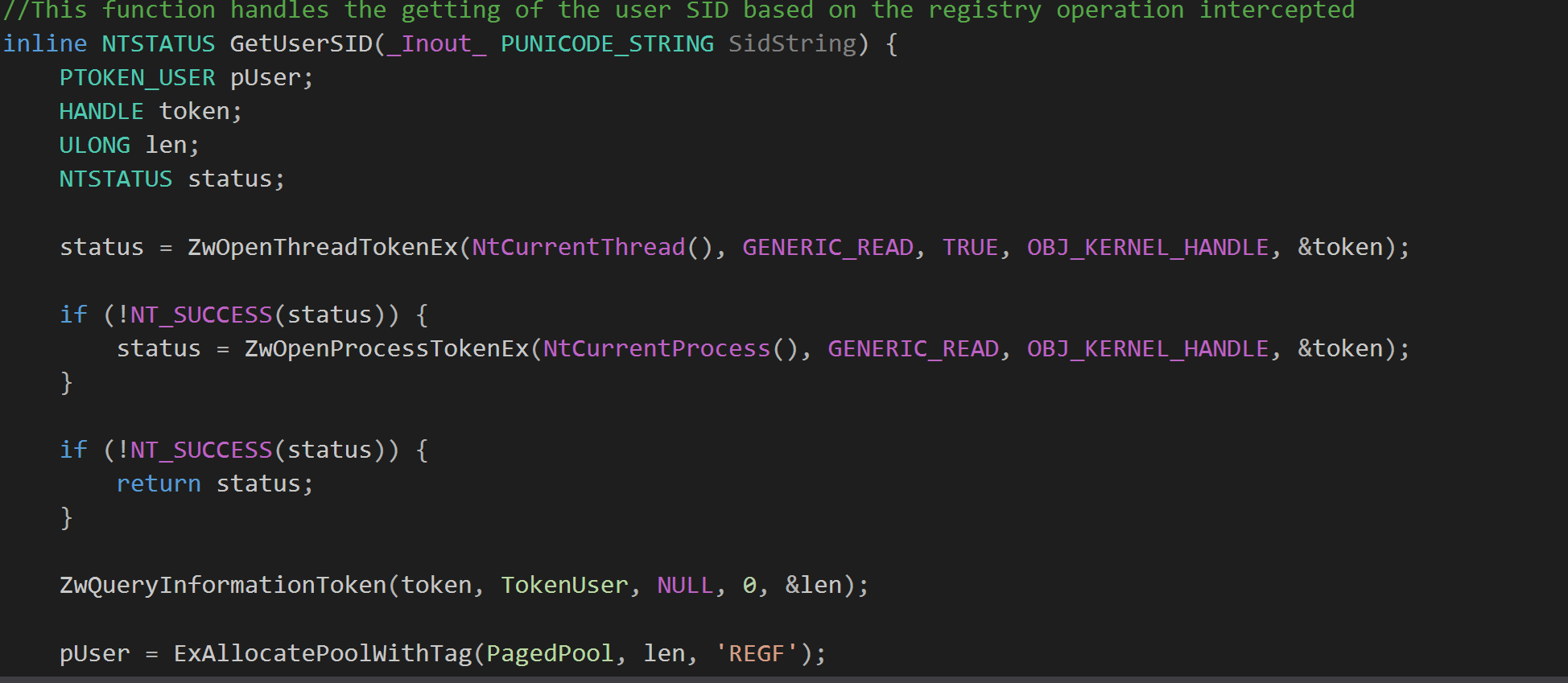


## Regfltr.h

* This header file handles the collection of timestamp and user SID information from the intercepted registry operations.
* Function “GetTimestamp()” returns the time of the registry operation in Unix Epoch time seconds as an unsigned long long.

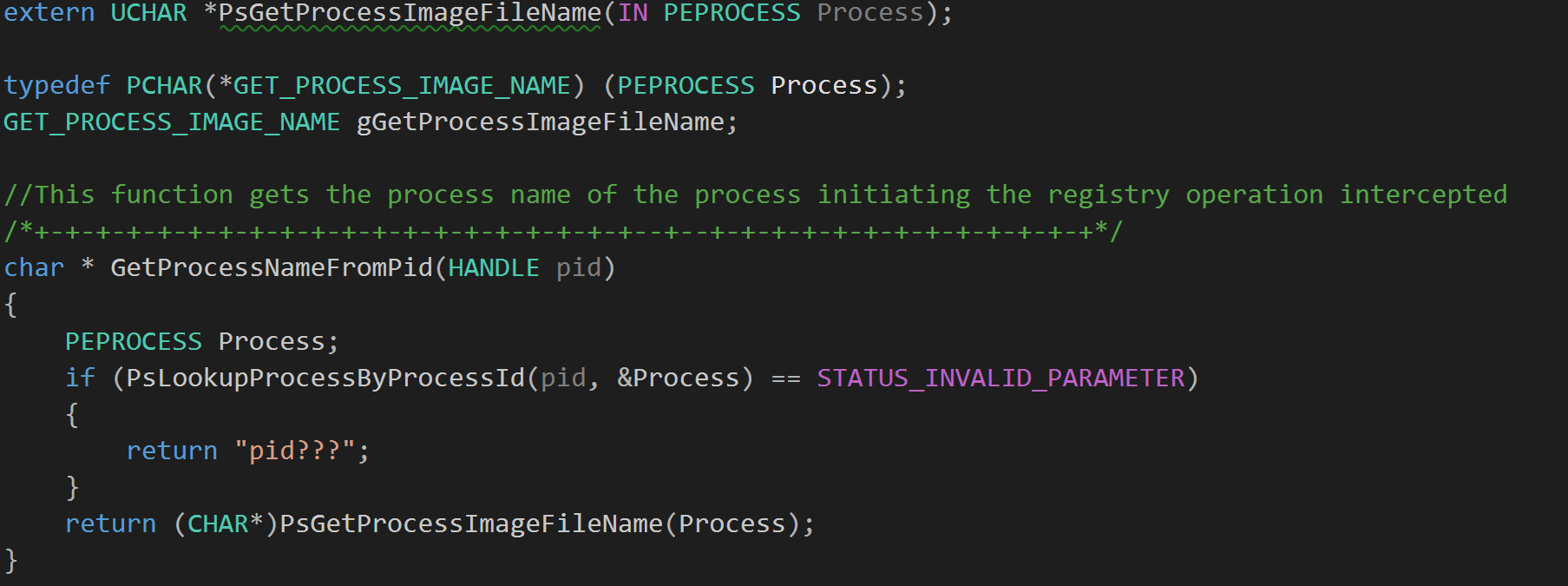


* Function “GetUserSID()” returns the SID of the user who initiated the registry operation based on the token associated with the thread initiating the registry operation.

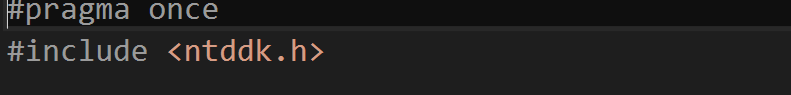


## Tools.h

* This header file handles the retrieval of the names of the processes that initiate intercepted registry operations.
* Function “GetProcessNameFromPid()” takes in the indentifying handle of the process that initiate the intercepted operations and returns the process name associated with that particular process ID.

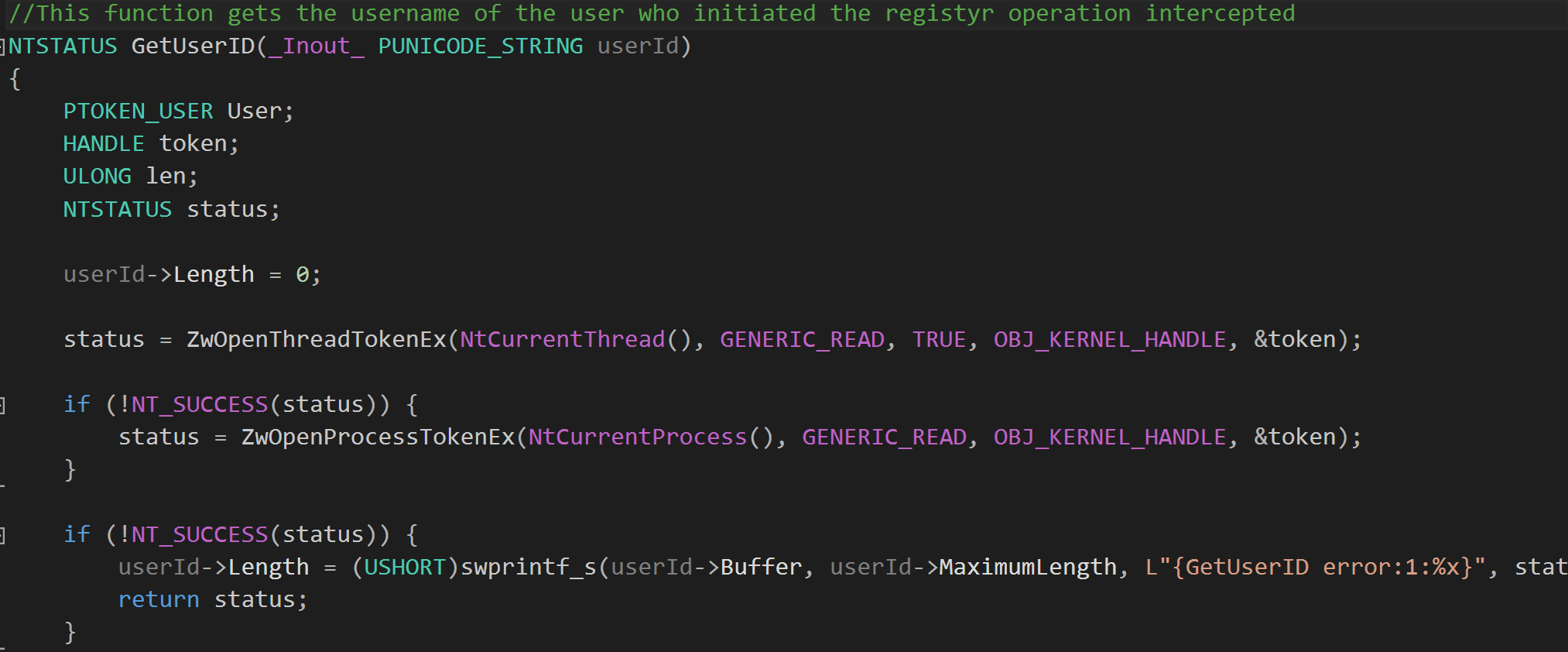


* The external dependency header file “ntddk.h” is not compatible with the header file “ntifs.h”, so any function that requires the “ntddk.h” file to be included should be preferably placed in the “tools.h” header file.

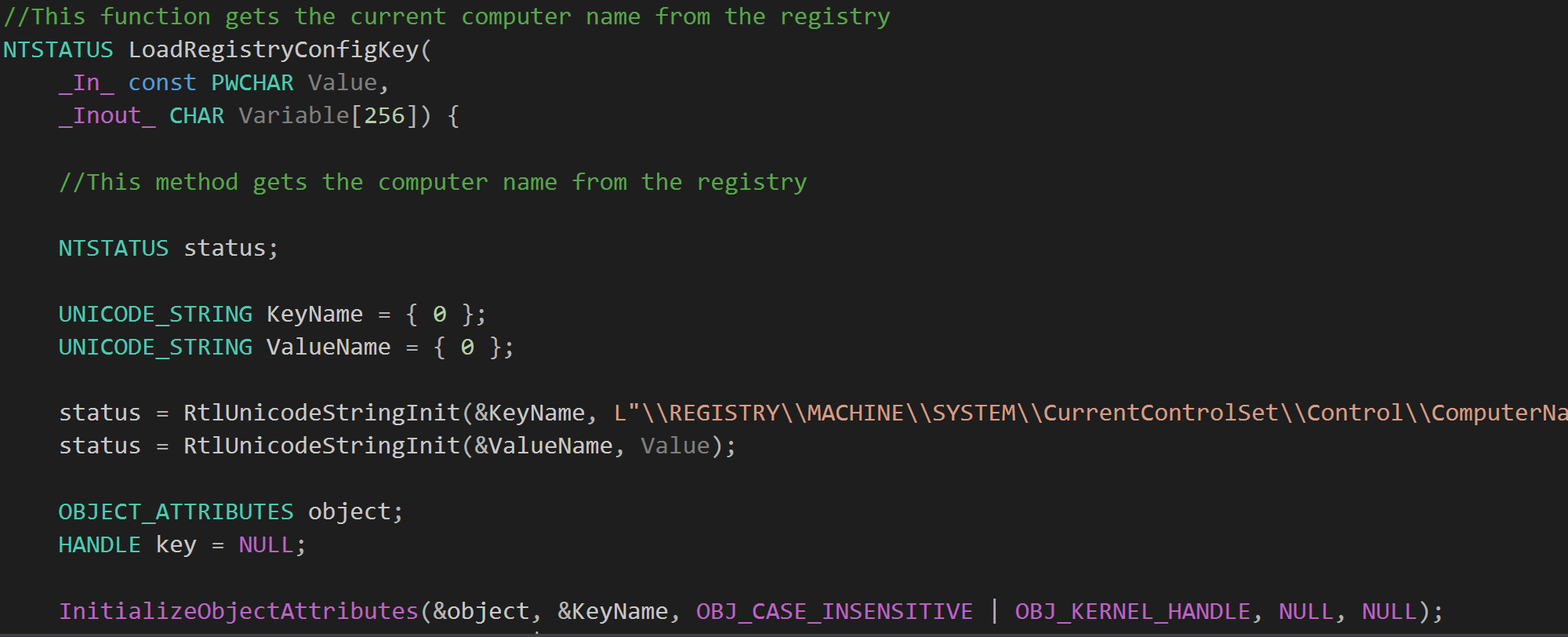


## Tool.c

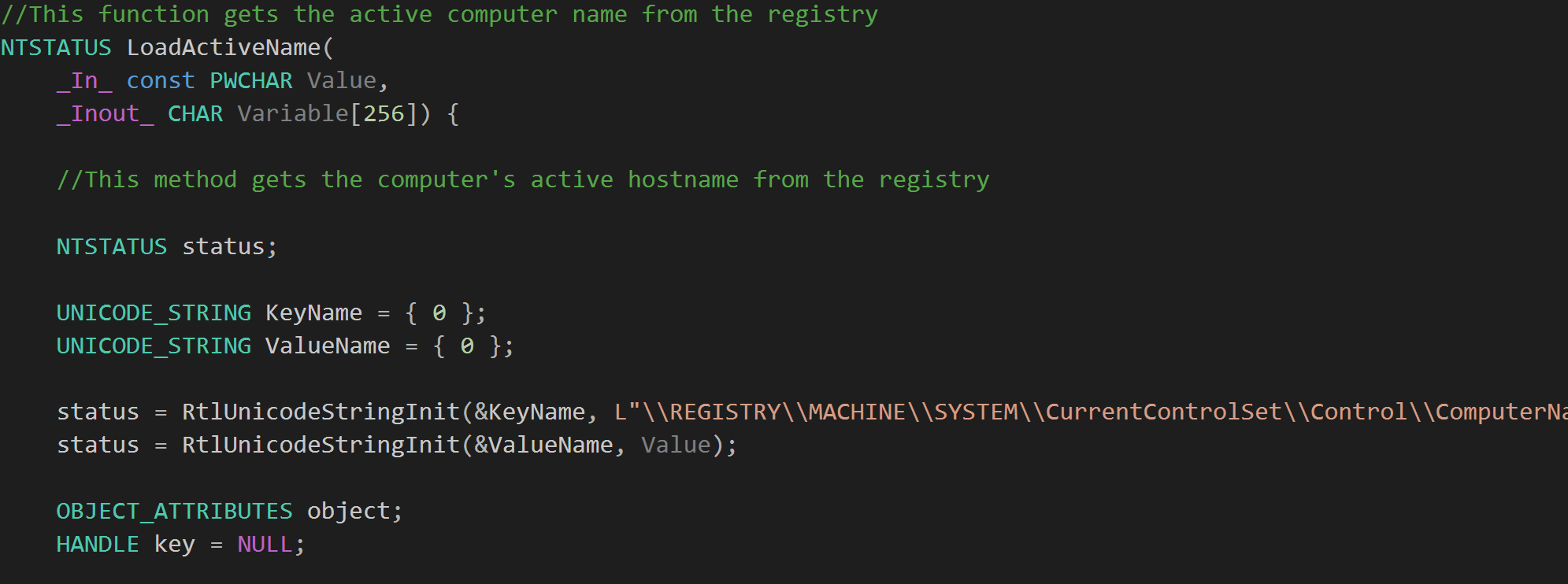
* This file handles the collection of username, current computer hostname and active computer hostname associated with the intercepted registry operation.
* Function “GetUserID()” retrieves and returns the username of the user that initiated the registry operation based on the token associated with the operation intercepted.



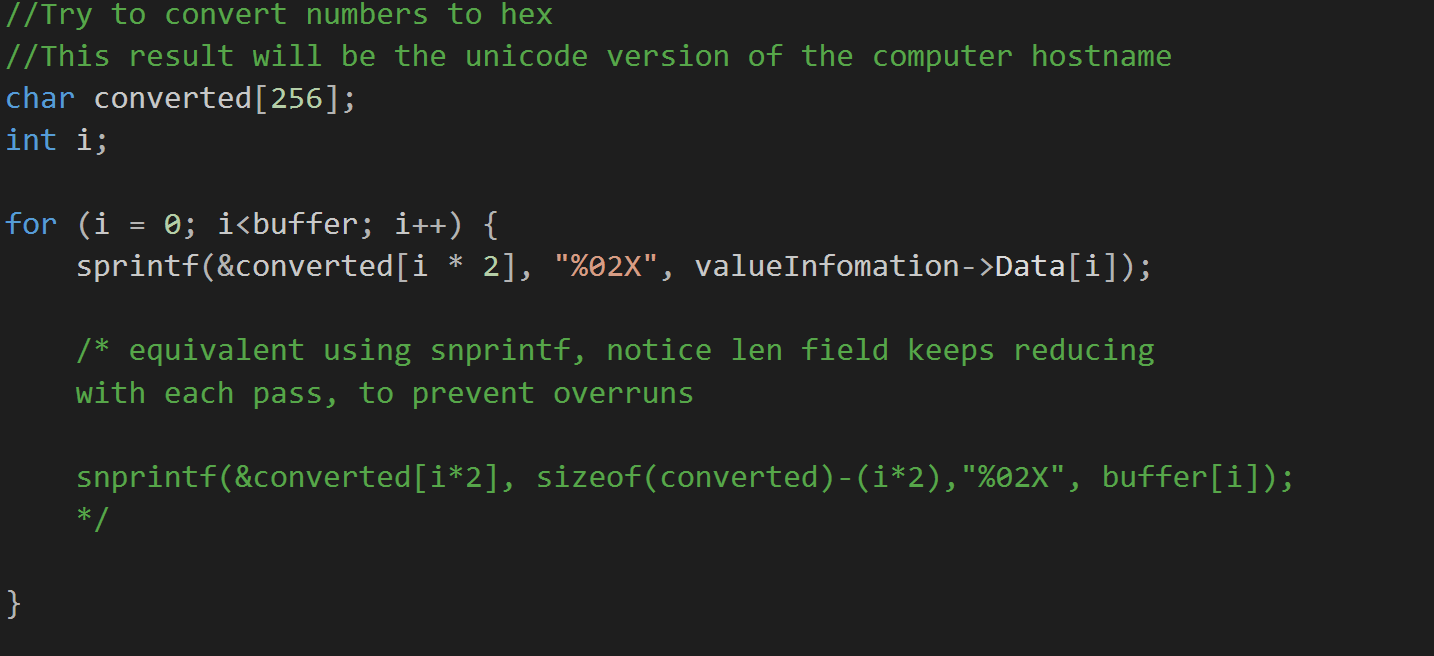
* Function “LoadRegistryConfigKey()” reads the value under “ComputerName” stored in the registry at “HKEY\_LOCAL\_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\ComputerName\\ComputerName” and returns it as the current computer hostname.



* Function “LoadActiveName()” reads the value under “ComputerName” stored in the registry at “HKEY\_LOCAL\_MACHINE\\SYSTEM\\CurrentControlSet\\Control\\ComputerName\\ActiveComputerName” and returns it as the active computer hostname.

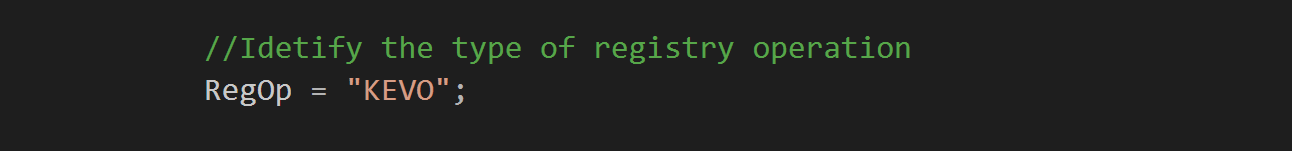


* Both the current computer hostname and active computer hostname are converted to Unicode before being returned as output of their functions.



## Acronyms Used to Identify Registry Operations

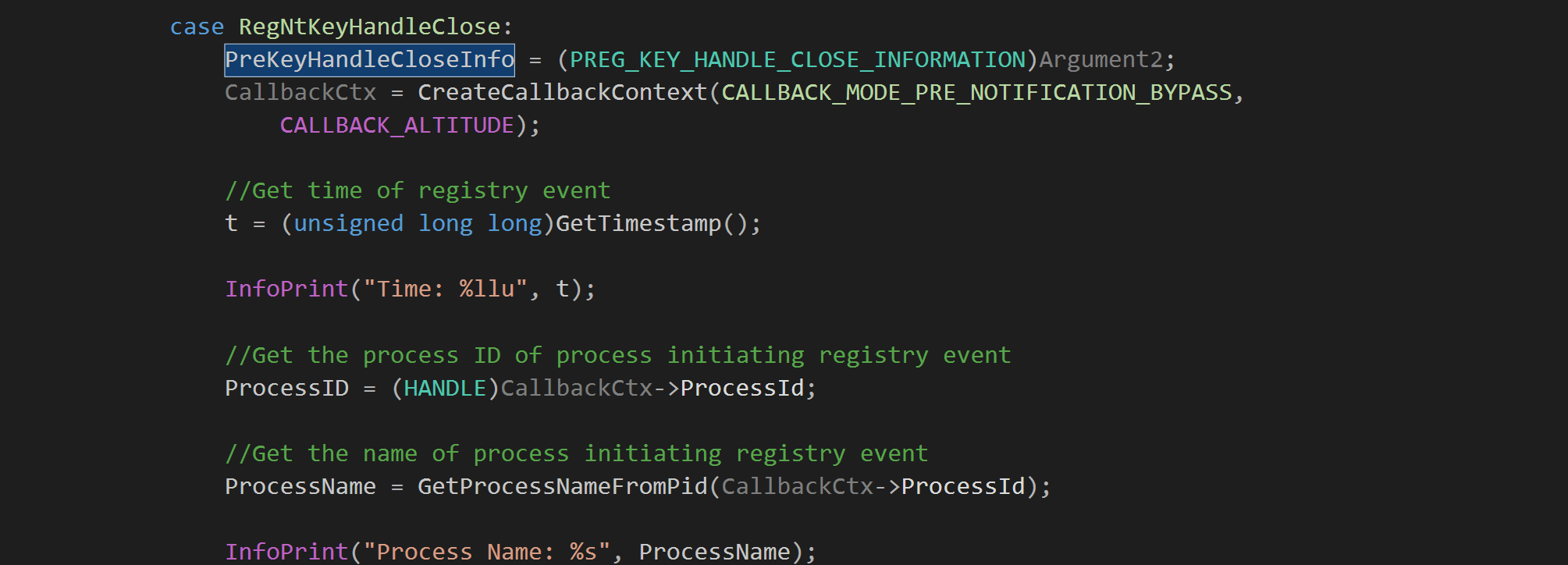
* For each registry operation intercepted, they are assigned a identifying acronym to define what type of registry operation it is.
* Below is a list of all the acronyms used and what they represent:
  + **KCO:** Key Creation Operation
  + **KVSO:** Key Value Setting Operation
  + **KVDO:** Key Value Deletion Operation
  + **KDO:** Key Deletion Operation
  + **KRO:** Key Renaming Operation
  + **KOO:** Key Opening Operation
  + **KVQO:** Key Value Querying Operation
  + **KQO:** Key Querying Operation
  + **KSSO:** Key Security Setting Operation
  + **KEO:** Key Enumeration Operation
  + **KEVO:** Key Value Enumeration Operation
  + **MKQO:** Multiple Key Querying Operation
  + **KISO:** Key Information Setting Operation



(Example registry operation type initialisation)

# Problem with Interception of Registry Operations

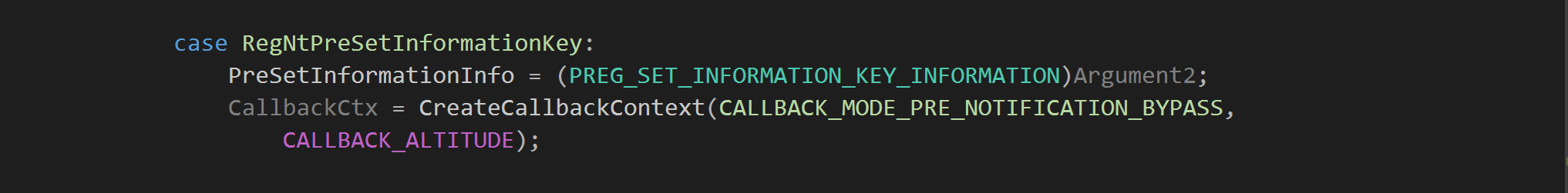
* There is a problem whereby when a new registry operation type case is made in the callback routine in regfltr/sys/Pre.c, the testing machine on which the filter driver is run will freeze and restart.
* A screenshot of an example of code which causes the problem is show below:



* The screenshot shows the addition of new cases to extract information from more registry operation types beyond the existing 13. This is causing the freezing problem on the target machine running the filter driver

# Adding of Post-Notification Registry Operation Interception

* Post-notification operations should be intercepted as well
* Post-notification = After registry operations have taken place in the registry
* Currently only pre-notification operations are intercepted



* Notice in the screenshot above how the CallbackCtx (Callback Context) is set as “CALLBACK\_MODE\_PRE\_NOTIFICATION\_BYPASS”
* Maybe changing it to “CALLBACK\_MODE\_POST\_NOTIFICATION\_BYPASS” would allow for the interception of post-notification registry operations

**Running the Windows Registry Filter Driver**

In this section, we will be going through how to setup the connection between the Windows Registry Filter Driver to Redis as well as the Redis to Python.

**Windows Registry Filter Driver (log.h)**

The purpose of this class is to declare all the methods that are being used in the log.c class.

**Windows Registry Filter Driver (log.c)**

The purpose of this class is to establish a connection with Redis from the Windows Registry Filter Driver and send data to Redis after taking in and formatting the information captured from each registry operations that are being intercepted. The method used is using a Winsock Kernel to connect to Redis through a socket with its port and network address. Below provides the relevant links needed to understand how it works. Communication through the socket also uses the Redis Protocol which further information can be found in the link under the Redis section.

<https://docs.microsoft.com/en-us/windows-hardware/drivers/network/introduction-to-winsock-kernel> (Introduction to Winsock Kernel)

**Redis**

Redis is a key value database which is most likely more optimized for Linux systems. However, you can still get it setup in a Windows environment. Below provides the relevant links needed to get it setup on your Windows environment and understand what it does

<https://redis.io/> (What is Redis?)

<https://github.com/rgl/redis/downloads> (Setup for Redis Windows)

<https://redis.io/topics/rediscli> (Redis Commands)

<https://redis.io/topics/protocol> (Redis Protocol)

**redis\_regfltr.py**

The main purpose of this Python file is

* To check and retrieve data that is being logged into Redis
* To convert any hex value of *hostname* and *activename* to readable format so that users can easily understand the data
* To print and write out the log onto an external CSV file.

The CSV will be created and written in the same location of the Python file itself.

Before using this file, ensure that the Redis Python is successfully setup on the computer. Below shows the relevant links on how to get it setup on your computer. More information on how the codes of the Python file works can be found in the comments in the file itself.

<https://redislabs.com/lp/python-redis/> (Setting up Redis Python)

Note that if you need to have Python installed if you want to use the *pip* command.

<https://www.python.org/downloads/> (Downloading Python)

The platform I used to run the Python script is PyCharm

<https://www.jetbrains.com/pycharm/> (PyCharm)