**SWINBURNE UNIVERSITY OF TECHNOLOGY**

School of Science, Computing, and Engineering Technologies

Text

Description automatically generated

Malware Analysis

COS20030

**Assignment 1**

Practical Application

**TUTOR**

Harry Ouaida

**STUDENTS**

Tran Anh Thu Pham - 103818400

Word count: 1997

**HAWTHORN – JANUARY 2025**

1. **Section 1 - Q1.exe (55 points).**

*In this section, you have a sample of a passive backdoor malware. A passive backdoor does not reach out to a command-and-control server; instead, it waits for a connection to be made to it. More specifically, a passive backdoor starts listening on a port for incoming connections. When a connection is made to it, it starts communicating and receiving commands to perform different actions on the infected machine.*

*Our goals in this exercise are:*

*i. Perform basic static analysis to find out more about the malware file:*

1. *Confirm that it’s packed and recognise the packer.*
2. *Find the MD5 hash and Entropy of the packed file’s PE header*
3. *Find when the file was compiled*

*ii. Perform dynamic analysis to:*

1. *Find out which port number the backdoor listens on*
2. *Use the Netcat (ncat) tool to emulate the server contacting and communicating the backdoor*
3. *Try some of the features of the backdoor by sending specific commands to the backdoor and observing its malicious behaviour*

*iii. The final goal is to unpack the backdoor sample using a debugger and a plugin to dump the unpacked code and reconstruct its headers.*

*The file to analyse is Md5: 8f7e26469d4a00136c0b76e6249ce684*

1. **Use q1.exe for this exercise. List three indicators that q1.exe is packed. What packer was used to pack q1.exe? (5 points)**

To view the indicator that q1.exe is packed, I used DiE tool. Firstly, I opened the tool via its desktop shortcut and loaded the q1.exe file.

After that, I checked the advance box to see more file info.

The following are indicators that the q1.exe file is packed (Fig 1, 2, 3):

* File/Section **entropy > 7** (Click “Entropy”)
* File with **status “packed”**
* **Unusual section names**: UPX that indicates that it is “packed”

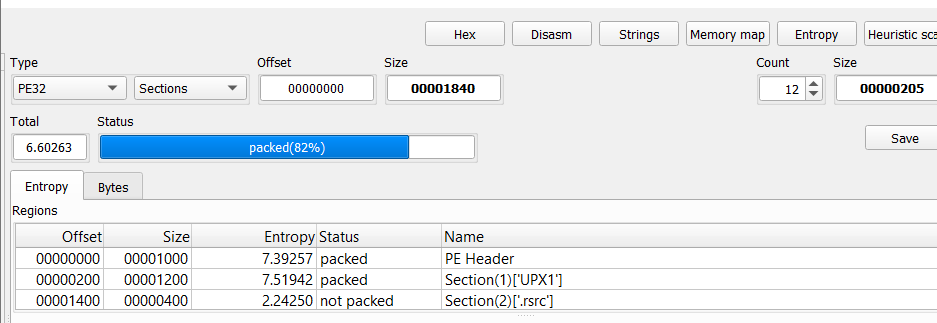




Fig 1. DiE Entropy section

* Sections with **unusual raw/virtual sizes**

Click “Section” to view the following info:

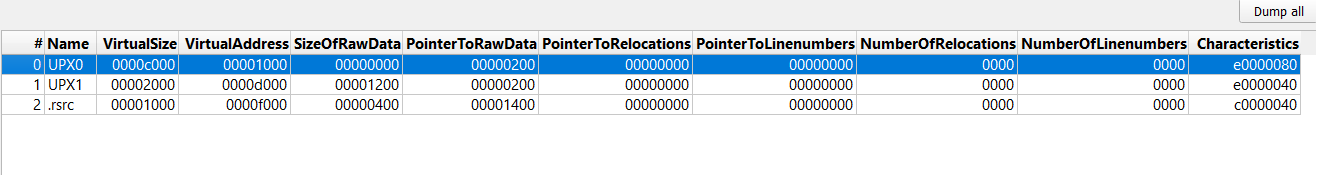


Fig 2. Overview of section’s virtual sizes in DiE

* The packer used to pack q1.exe is **UPX (3.95)[NRV, brute]**. I opened the file in DiE to inspect the headers and see the characteristics of the file. It immediately appears that the tool recognizes the packer used for packing this file as “Packer: UPX”.

A close up of a computer

Description automatically generated

Fig 3. Overview of packer’s model in DiE

1. **Answer the following questions:**
2. **What is the MD5 hash of the packed file’s PE Header? (2 points)**

To view the MD5 hash of the packed file’s PE Header, I went to the file info in the DiE tool and then went to the “Hash” section (Fig 4).

MD5 hash: **ebf55c5acad3066eb25a291faf3b4086**

**A screenshot of a computer

Description automatically generated**

Fig 4. Hash of PE Header

1. **What is the entropy value of the packed file’s PE Header? How did you find this? (3 points)**

Entropy value of the packed file’s PE Header: **7.39257**

To find this, I loaded the file q1.exe to the DiE application and then clicked the “Entropy” section to view the entropy value of the packed file’s PE Header

(Fig 5).

A screenshot of a computer

Description automatically generated

Fig 5. Entropy value of PE Header

1. **On what date was this program compiled? (2 points)**

Choose the "IMAGE\_FILE\_HEADER” subsection in the IMAGE\_NT\_HEADERS section, to view the TimeDateStamp which is the date this program compiled (Fig 6): **2001-11-21 04:17:53**

**A screenshot of a computer

Description automatically generated**

Fig 6. TimeDateStamp in IMAGE\_FILE\_HEADER section

1. **Execute the program and find out what port q1.exe listens on. How did you find this information? (5 points)**

Firstly, I opened the” Process Explore” tool. In the list of processes, find the q1.exe which is in the “procexp.exe” directory and below the “procexp64.exe”. Click the “q1.exe” process, and the window showing the properties of the q1.exe process will show up. Next, choose the TCP/IP tab to view the port number the q1.exe program is listening on. The port number is under the Local Address column (Fig 7).

Port number: **5277**

**A screenshot of a computer

Description automatically generated**

Fig 7. q1.exe process’s property

1. **While the program is running, open a CMD window then type the following command and use the port number you found in the previous question. The ncat tool will connect to the port number specified and start communicating on this port. This should result in ncat talking to the q1.exe process.**

**ncat localhost <port number>**

1. **Once you have run the ncat command, type “?” to list the different commands that the q1.exe can execute. Provide a screenshot of the list of commands. (3 points)**

Here is the screenshot I took after running the “ncat localhost 5277” command and typing “?” in the command prompt (Fig 8).

**A screenshot of a computer

Description automatically generated**

Fig 8. ncat command and help command

1. **Use the correct command to download a file from the URL “http://www.maldomain.com" with q1.exe. This operation will fail due to not being able to contact the real website, however, the executable scans the registry for the status of multiple keys relating to network connectivity. One of those keys relates to [PunyCodeLinks to an external site.](https://www.jamf.com/blog/punycode-attacks/" \t "_blank). What application does q1.exe check to see if PunyCode is enabled? How do you know? (7 points)**

q1.exe examines **Internet Explorer application** settings to check if PunyCode is enabled. To analyse this, I used Process Monitor.

Firstly, I launched the Process Monitor tool and applied the following filters (Fig 9):

* Process name – is – q1.exe
* Path – contains – Punycode

**A screenshot of a computer

Description automatically generated**

Fig 9. Filter application to check if Punycode is enabled

After configuring Process Monitor, I typed the following commands in the command prompt:

* ncat localhost 5277
* url https://www.maldomain.com

The figure below displays the processes captured during this process. Notably, the “Process Name” is “q1.exe”, and the “Path” includes "EnablePunycode".

Additionally, the registry keys in the “Path” column contain "Internet Settings" just before "EnablePunycode." This indicates that the "EnablePunycode" setting is used by Internet Explorer to determine whether Punycode URLs are displayed in their original encoded form or decoded into a readable Unicode format (Fig 10).

**A close-up of a computer screen

Description automatically generated**

Fig 10. Processes captured after applying specific filters

1. **Through ncat, send the “i” command to make q1.exe install itself. What is the registry key that the malware uses for persistence? What is the full path of the file that is made persistent? How did you find this information? (10 points)**

* The registry key that the malware used for persistence: **HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Run\bndshell**
* The full path of the file that is made persistent: **C:\Users\student\COS20030 Assignment 1\midsemester\_assignment\midsemester\_assignment\q1.exe**

To find this information, I used ncat with the “i” command and configured Process Monitor with two filters:

* Process name – is – q1.exe
* Operation – is – RegSetValue

The “RegSetValue” operation captures actions where a process creates or modifies values in the Windows Registry.

After applying the filters, a process was displayed. I double-clicked it to view detailed information, revealing the registry key used for persistence and the file's full path (Fig 11).

A screenshot of a computer

Description automatically generated

Fig 11. Registry key used for persistence and file's full path

1. **Unpack the file, record a video of yourself doing it, and submit both the unpacked file and the video. (3 points for the file and 12 points for the video)**

* Jump instruction address: 0040e108
* Jump to address: 004011cb
* Youtube link: <https://www.youtube.com/watch?v=1ViKgoT3Spw>

1. **Find out what API is used by this malware to download files from the given URL. (3 points)**

In the DiE tool, I went to the “Strings” section and applied the filter: “URL” (Fig 12).

API used to download files: **URLDownloadToFileA**

**A screenshot of a computer

Description automatically generated**

Fig 12. API used by malware to download files

1. **Section 2 - Q2.exe (40 points)**

*In this section, you have a sample of a different malware. We want to learn about some of the characteristics of this file and understand some of the features of this malware by examining its code in a disassembler.*

*Our goals in this exercise are:*

* *Use the IDA Freeware tool to disassemble the file and find out about the libraries imported by this malware*
* *Find where a specific string is used in the code*
* *Analyze and understand some of the functions in the malware by examining the decompiled code (Pseudocode), renaming function names and variables to meaningful names and adding comments in the code to explain what the function does.*
* *Analyse and understand a part of the disassembled code (Assembly code) of the program by reading the assembly instructions between two given addresses in the code and adding comments in the disassembled code to explain what the code does.*

#### File to analyse: Md5: 60b12ad2f23f2e4a594daa17cb8f517c

1. **How many DLL files does q2.exe import functions from? (3 points).**

I ran DiE tool and uploaded the q2.exe file into it then I went to the “import” section to view the dll files.

It imports 4 dll files:

* **KERNEL32.dll**
* **ntdll.dll**
* **RPCRT4.dll**
* **SHELL32.dll**

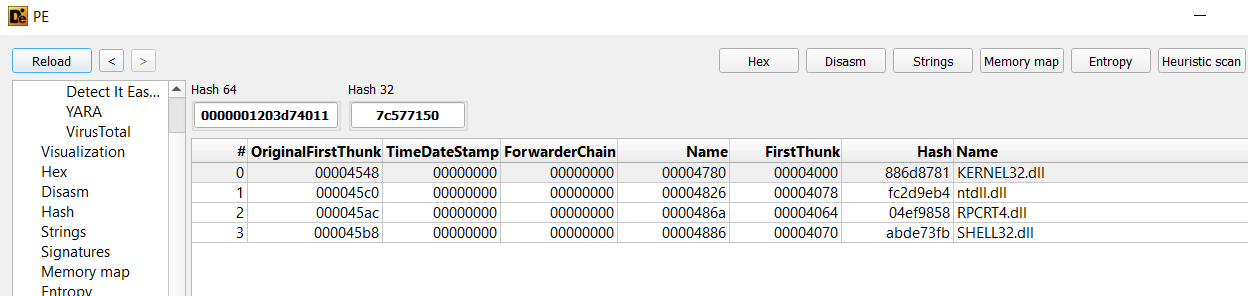




Fig 13. DLLs q2.exe used to import functions

1. **What’s the address of the string “ComSpec”? (3 points).**

I ran IDA tool and uploaded the q2.exe file into it. To see the list of strings, navigate to View then click “Open subviews”. Next, select “Strings” or use Shift+F12.

The address of the string “Comspec”: **.rdata:00404448**

A screenshot of a computer

Description automatically generated

Fig 14. List of string’s addresses in IDA tool

1. **How many arguments are passed to FUN\_00401d30? (3 points).**

**One argument** is passed to this function which is “LPCSTR param\_1”.

To find this information, I used Ghidra to upload the q2.exe file. After that, I click on the function name in the Symbol Tree to view the decompiled codes.

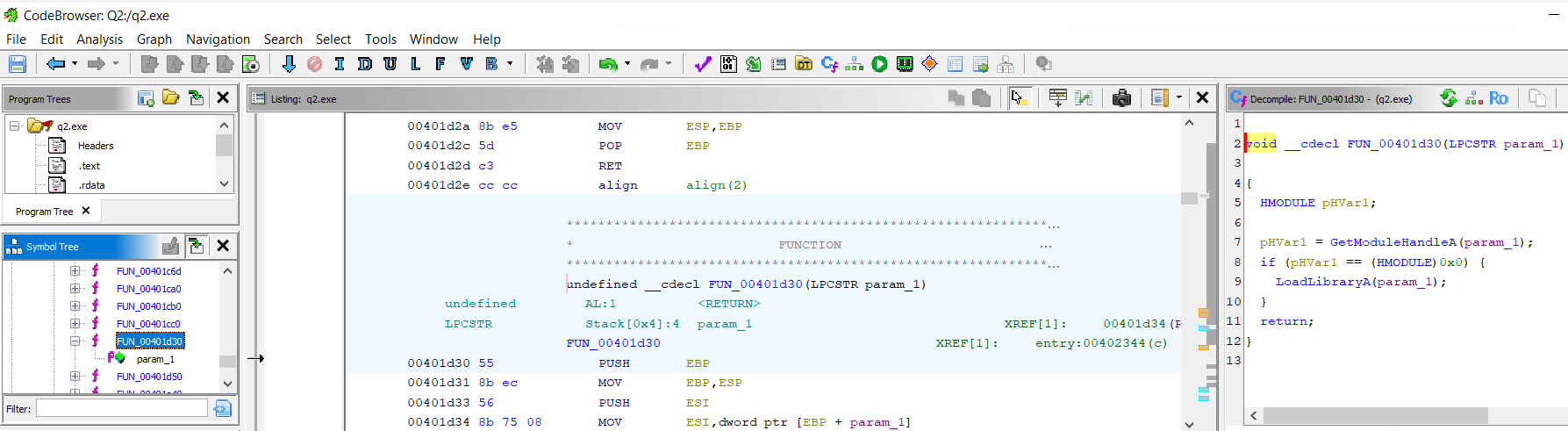


Fig 15. Argument passed to FUN\_00401d30

1. **What undocumented functions from NTDLL.DLL are used in this program? (6 points).**

Undocumented functions in NTDLL are native functions that provide low-level access to the kernel and system services. To find these native functions, I loaded the q2.exe file to the IDA tool and then went to the “Import” tab. These functions are in the ntdll directory (Fig 16). The following are undocumented functions used in this program:

* **NtTerminateProcess**: Terminates a process [1]
* **NtQueryVirtualMemory**: Retrieves information about a region of virtual memory in a process [1]
* **NtProtectVirualMemory**: Changes the protections on a region of virtual memory in a process [1]
* **RtlCompareMemory**: Compares two blocks of memory [1]

The listed functions above are undocumented functions (native functions) because they reside in NTDLL.DLL and start with Nt or Rtl prefixes. These functions are designed for low-level system interactions such as memory management and process or thread creation that might require direct system calls to the kernel.

**A screenshot of a computer

Description automatically generated**

Fig 16. Examine Undocumented functions from ntdll.dll using IDA

1. **Rename param\_1 in FUN\_00401d30 to something more descriptive. Justify the variable name you chose and save your work in IDA (7 points).**

In function FUN\_00401d30, the parameter was originally named lpModuleName. Since the parameter is specifically used for importing functionality from ntdll.dll, I renamed it to “**Import\_NTDLL**” to better reflect its purpose and origin. To rename the param\_1, click on the parameter through IDA's interface, press the 'n' key, and enter the new name.

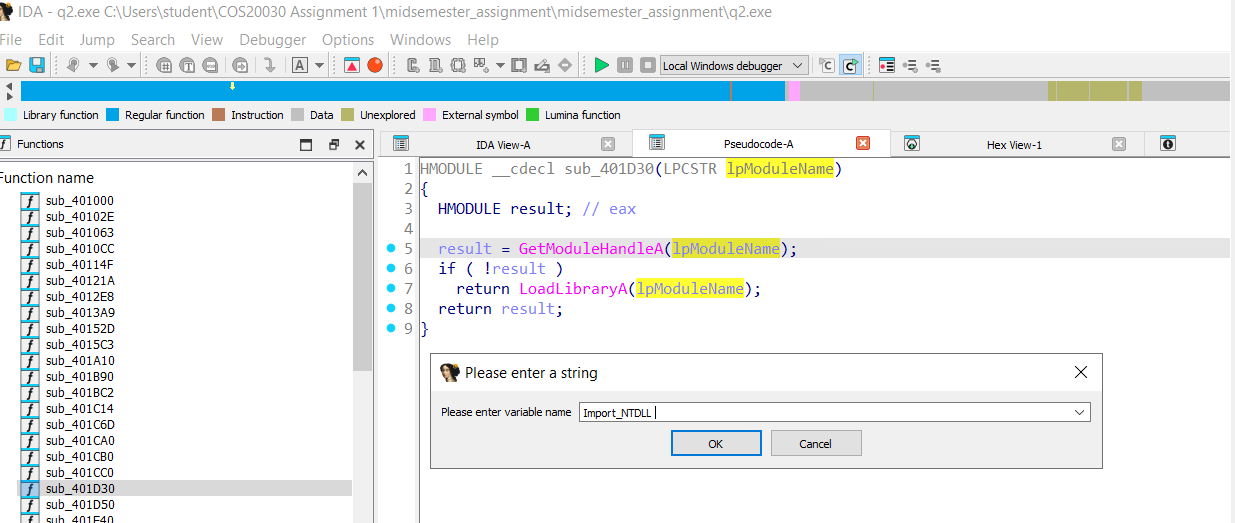


Fig 17. Rename the argument of the function

1. **Describe what the code between 0x0040239d and 0x004023b9 does and add comments in IDA. (8 points).**

The code between 0x0040239d and 0x004023b9 does two main things:

* First, it stores the address of the Wow64Process variable (from the stack) in the EDX register and pushes it as a parameter to call GetCurrentProcess and IsWow64Process [2].
* After calling IsWow64Process, it checks the value stores in Wow64Process. If this value is non-zero (meaning the process is running under WOW64), the function returns 1 [2].

**A screenshot of a computer

Description automatically generated**

Fig 18. Codes between 0x0040239d and 0x004023b9

To add comments in IDA, I press ‘;’ or ‘/’ in the disassembly (IDA View).

A screenshot of a computer

Description automatically generated

Fig 19. Adding comments for codes in the IDA disassembly view

Here is the code snippet after adding the comments to highlight the functionalities of each line of code:

A screen shot of a computer program

Description automatically generated

Fig 20. Code snippet after adding the comments

1. **Briefly describe what the function located at 402090 does. Suggest a descriptive name for that function. Change the function's name, add comments in the code and save your work in IDA (10 points).**

**A screen shot of a computer

Description automatically generated**

Fig 21. Subroutine Details and Code Structure at Address 0x00402090

**A screenshot of a computer program

Description automatically generated**

Fig 22. Decompiled Subroutine Showing Command Execution Logic of function located at 402090

The main purpose of the function located at 402090 is to delete its own executable file while the process is still running. It works by getting the executable’s path using GetModuleFileNameA and converting it to a short path format with GetShortPathNameA. After that, it creates a command to delete the file (/c del <short\_path> >> NUL) and gets the command processor path through GetEnvironmentVariableA. Lastly, it runs the delete command using ShellExecuteA.

I changed the function name from sub\_402090 to **DeleteCurrentProcessFile** because it directly describes what the function does – it deletes the current process’s executable file as described above.

To rename the function,I clicked on the function name and press ‘n’.

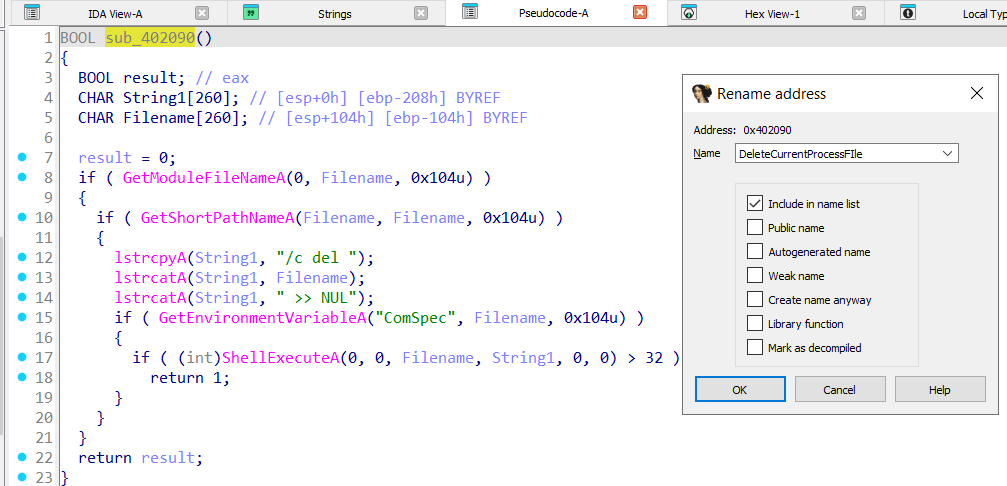
****

Fig 23. Rename the function located at 402090

Here is the code after adding the comments:

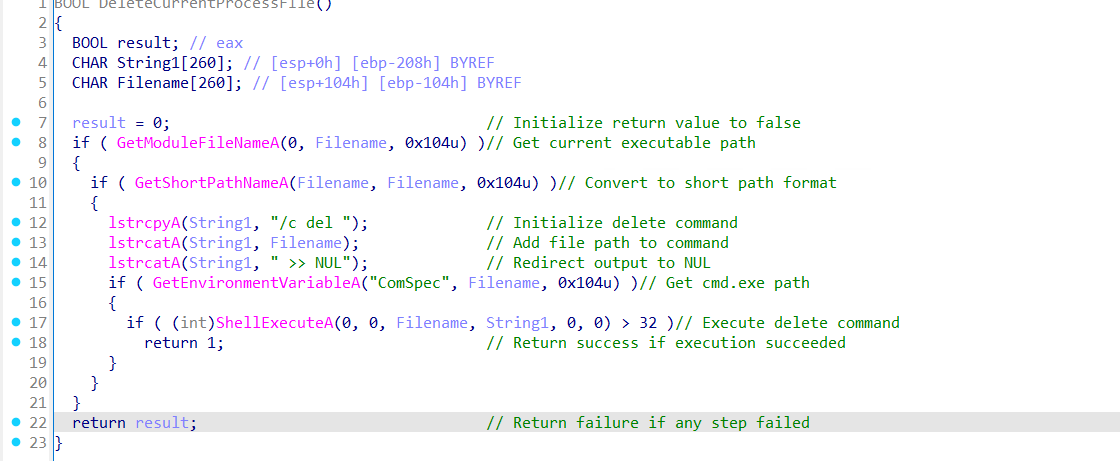


Fig 24. Code snippet after adding the comments

1. **Reflection**

When doing this assignment, I felt challenged in the following parts:

* 1. **Finding the jump to the OEP using the Ghidra tool**

One of the key challenges I encountered during this assignment was finding the jump to the OEP (Original Entry Point) using the Ghidra tool. Despite practising with Ghidra in lab sessions, I found applying the tool effectively in this context challenging. The primary difficulty stemmed from navigating Ghidra's graphical user interface (GUI) and understanding the steps required to locate the JUMP instruction's address, identify its destination address, and analyse how the code operates. To address this challenge, I have reviewed the lab 5 paper and followed the important steps listed in this lab.

* 1. **Research the purpose of code snippets and explain them**

Another significant challenge I encountered was researching the purpose of code snippets and explaining them. This was the most demanding aspect of the assignment, as it required a deep understanding of the functionality of each line of code to provide accurate and descriptive comments. For instance, in question 6 section 2, I have to describe the functionalities of the code block between 0x0040239d and 0x004023b9. To tackle this, I researched the paper on the Windows API documentation and assembly language references extensively. I focused particularly on understanding the WOW64 system and its related API functions like GetCurrentProcess() and IsWow64Process(). The references to these documents are listed in the **Reference** section below.

1. **Reference**
2. *Native API Functions*. (2016). Geoffchappell.com. <https://www.geoffchappell.com/studies/windows/win32/ntdll/api/native.htm>
3. GrantMeStrength. (2022, November 18). IsWow64Process function (wow64apiset.h) - Win32 apps. Microsoft.com. <https://learn.microsoft.com/en-us/windows/win32/api/wow64apiset/nf-wow64apiset-iswow64process>