SPL-1 Project Report, [2022]

[Malware Detecting system]

[SE 305]: [Software Project Lab]

Submitted by

[*Tanjuma Tabassum Jerin*]

BSSE Roll No. : 1312

BSSE Session: 2020-2021

Supervised by

[Moumita Asad]

Designation: Lecturer
Institute of Information Technology

Institute of Information Technology
University of D
[21-05-2023]

Table of Contents

1.Introduction:

The "Malware Detecting System" is a project aimed at providing an automated solution for identifying and detecting malware in executable (exe) files. Malware, short for malicious software, refers to any software specifically designed to cause harm, compromise security, or disrupt the normal functioning of computer systems. This can include viruses, worms, Trojans, ransomware, and other types of malicious code.

The system utilizes various techniques such as SHA256 hashing, string matching algorithms (trie), and PE header parsing to determine the presence of malware. By analyzing the characteristics and behavior of executable files, the system can identify potential threats and take appropriate actions to mitigate them.

1.1 Background Study:

Malware poses a significant threat to computer systems and networks. With the increasing sophistication and variety of malware, traditional signature-based detection methods alone are no longer sufficient. Advanced techniques and algorithms are required to accurately detect and mitigate malware infections. This project aims to address this need by implementing an efficient and effective malware detecting system.

For doing this project I had to implement a hashing algorithm named SHA 256 , a string matching algorithm for matching the malware's hash value with the known malware hash values. PE header parsing of the exe file also had to done. And also a encoding of a exe file.

The main 4 parts of the project are-

1. SHA 256
   1.input formatting
   2.hash buffer initialization
   3.message digest

4.output

2. String matching algorithm (Trie)
   1.Insert string in a trie
   2.search a specific string in a trie

3. PE header parsing that includes
   1.DOS header parsing
   2.File header parsing
   3.Optional header parsing
   4.Section header parsing

4. Encoding of a exe file
   Using bit shifting method

1.2 Challenges

Developing a malware detecting system comes with several challenges that require careful consideration and effective solutions. Some of the key challenges in this domain include:

- Malware Variability: Malware authors constantly evolve their techniques to evade

detection by security systems. They employ various obfuscation, encryption, and polymorphism techniques to make their malware harder to identify. To build an effective malware detecting system, it is crucial to stay updated with the latest malware variants and develop detection mechanisms that can handle the ever-changing landscape of malware. Considering this reason I decided to conduct a static analysis “PE header parsing” that parse the necessary information of the pe file and check the characteristics of the file that if it is malware or not.

- Performance: Malware detection involves analyzing large volumes of data, including

files, system behavior. This process can be computationally intensive and resource-consuming. It is important to design efficient algorithms and utilize optimized data structures to ensure the system can handle the processing requirements without significant delays or performance degradation.

- User Interface: While the technical aspects of a malware detecting system are crucial, the user interface plays a significant role in its usability and effectiveness. Designing a user-friendly interface that provides clear information about detected threats, offers appropriate actions to users, and facilitates intuitive navigation and configuration is essential. It ensures that users can effectively interact with the system, make informed decisions, and take necessary steps to mitigate identified malware threats.

- Converting input string inSHA256: Converting decimal to binary , binary to hexadecimal for ,decimal to hexadecimal is quite difficult in evaluating hash value. Implementing these part in c++ took too much time.

- Learning about PE file format: PE file format is a file format for executables in windows operating system. A PE file is a data structure that holds information necessary for the OS loader to load the file into memory and execute it. Understanding the full file format was a bit difficult.

- Parsing section header of a PE file: Parsing section header and evaluating the section names seemed the most difficult part for the project as there’s no uses of any extra library function that easily evaluate and parse the PE header.

- Limited Resources: Finding comprehensive and reliable resources online was a challenge. This scarcity hindered progress and required resourcefulness to explore alternative avenues for acquiring the necessary knowledge and guidance.

Had to study about all of the part of my project cause i had no idea of this project. And overcome the difficulties by analysing the part more and more and took help from various resources.

2.Project Overview:

The project architecture consists of the following main parts:

- File Input Module
- Hash value evaluating module
- Hash Matching Module
- PE Header Parsing Module
- User Interface Module
- File Management Module

Let’s discuss the functionalities of each part one by one-

1.SHA256 Hash value evaluation:

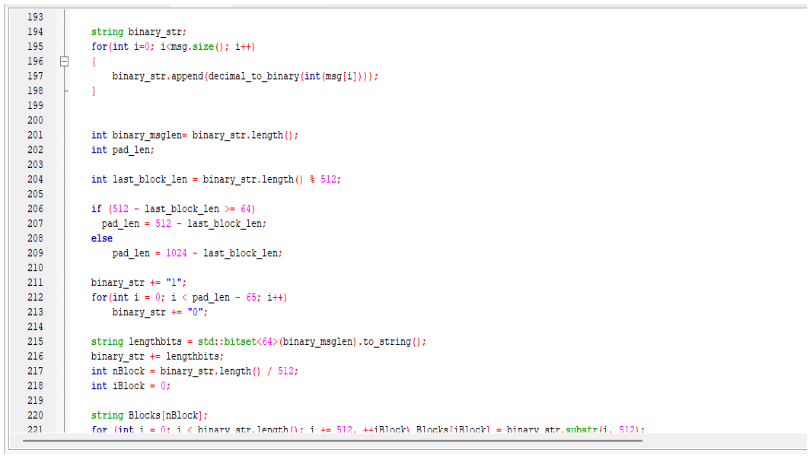
Calculates the SHA256 hash value of an executable file and compares it against a database of known malware hashes. SHA 256 has following steps-

- Input formatting

  Append padding bits

  Append length

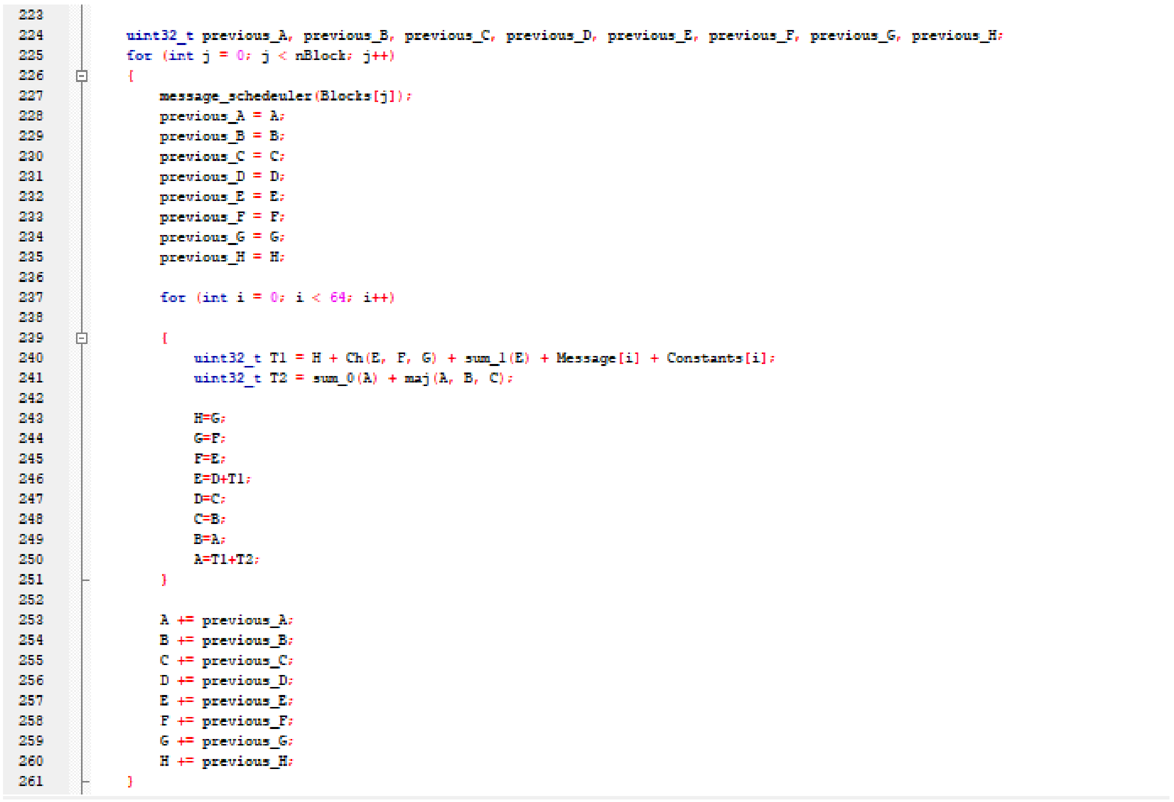
Here’s the input formatting part done in c++



```cpp

    string binary_str;
    for(int i=0; i<msg.size(); i++)
    {
        binary_str.append(decimal_to_binary(int(msg[i])));
    }


    int binary_msglen= binary_str.length();
    int pad_len;

    int last_block_len = binary_str.length() % 512;

    if (512 - last_block_len >= 64)
      pad_len = 512 - last_block_len;
    else
        pad_len = 1024 - last_block_len;

    binary_str += "1";
    for(int i = 0; i < pad_len - 65; i++)
        binary_str += "0";

    string lengthbits = std::bitset<64>(binary_msglen).to_string();
    binary_str += lengthbits;
    int nBlock = binary_str.length() / 512;
    int iBlock = 0;

    string Blocks[nBlock];
    for (int i = 0; i < binary_str.length(); i += 512, ++iBlock) Blocks[iBlock] = binary_str.substr(i, 512);
```

- Hash buffer initialization

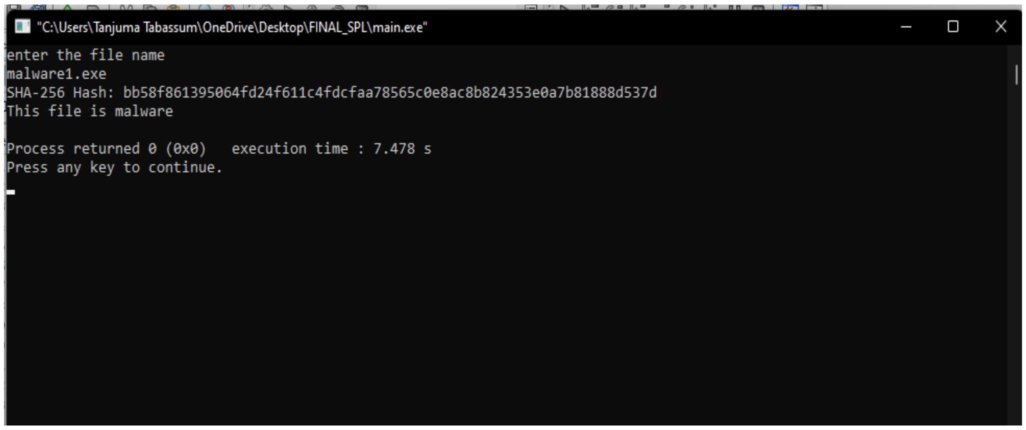
- Message processing



```cpp

    uint32_t previous_A, previous_B, previous_C, previous_D, previous_E, previous_F, previous_G, previous_H;
    for (int j = 0; j < nBlock; j++)
    {
        message_schedeuler(Blocks[j]);
        previous_A = A;
        previous_B = B;
        previous_C = C;
        previous_D = D;
        previous_E = E;
        previous_F = F;
        previous_G = G;
        previous_H = H;

        for (int i = 0; i < 64; i++)

        {
            uint32_t T1 = H + Ch(E, F, G) + sum_1(E) + Message[i] + Constants[i];
            uint32_t T2 = sum_0(A) + maj(A, B, C);

            H=G;
            G=F;
            F=E;
            E=D+T1;
            D=C;
            C=B;
            B=A;
            A=T1+T2;
        }

        A += previous_A;
        B += previous_B;
        C += previous_C;
        D += previous_D;
        E += previous_E;
        F += previous_F;
        G += previous_G;
        H += previous_H;
    }
```

- Output

In the following figure shows that if a file known malware that’s hash value is saved in known malware hashset, so the file is detected as malware just evaluating it’s hash value



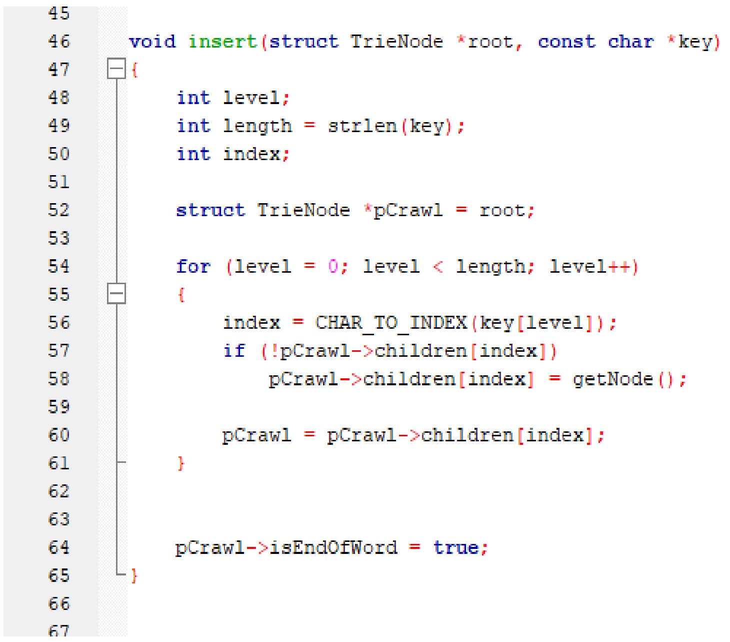
"C:\Users\Tanjuma Tabassum\OneDrive\Desktop\FINAL_SPL\main.exe"

```
enter the file name
malware1.exe
SHA-256 Hash: bb58f861395064fd24f611c4fdcfaa78565c0e8ac8b824353e0a7b81888d537d
This file is malware

Process returned 0 (0x0)   execution time : 7.478 s
Press any key to continue.
```

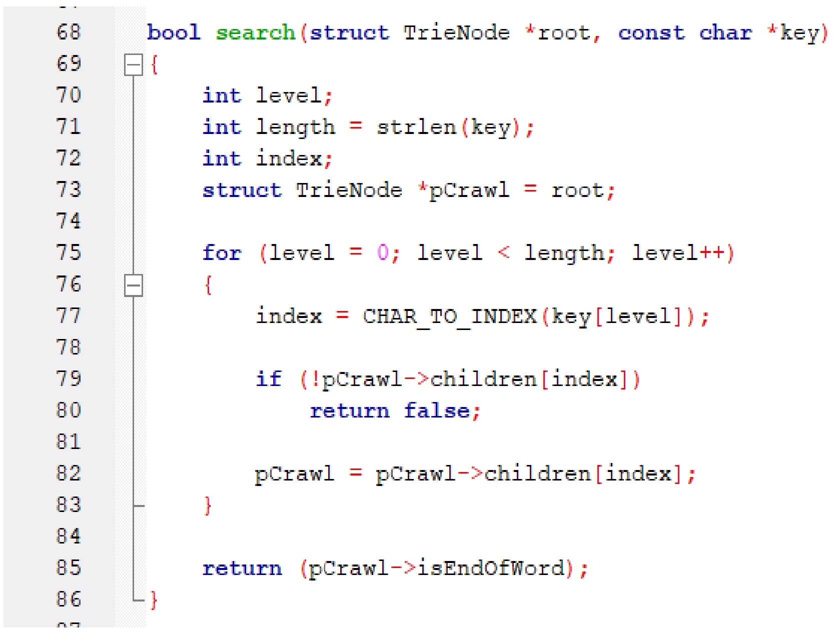
2. String Matching Algorithm:

Utilizes a trie-based string matching algorithm to efficiently search for matching hashes.Here i conduct two operation in trie. These are Insert and Search. I insert all the known hash values from file to the trie. Then conduct search operation in the inserted trie.

Search the hash value of the file that is evaluated by sha 256.



```c
void insert(struct TrieNode *root, const char *key)
{
    int level;
    int length = strlen(key);
    int index;

    struct TrieNode *pCrawl = root;

    for (level = 0; level < length; level++)
    {
        index = CHAR_TO_INDEX(key[level]);
        if (!pCrawl->children[index])
            pCrawl->children[index] = getNode();

        pCrawl = pCrawl->children[index];
    }


    pCrawl->isEndOfWord = true;
}
```



```c
bool search(struct TrieNode *root, const char *key)
{
    int level;
    int length = strlen(key);
    int index;
    struct TrieNode *pCrawl = root;

    for (level = 0; level < length; level++)
    {
        index = CHAR_TO_INDEX(key[level]);

        if (!pCrawl->children[index])
            return false;

        pCrawl = pCrawl->children[index];
    }

    return (pCrawl->isEndOfWord);
}
```

3.PE Header Analysis: if the input file is not detected as malware ,then for further test i conduct a static analysis ‘PE header parsing’ of the file to extract important information such as

1.number of initialized data

2. Major image version

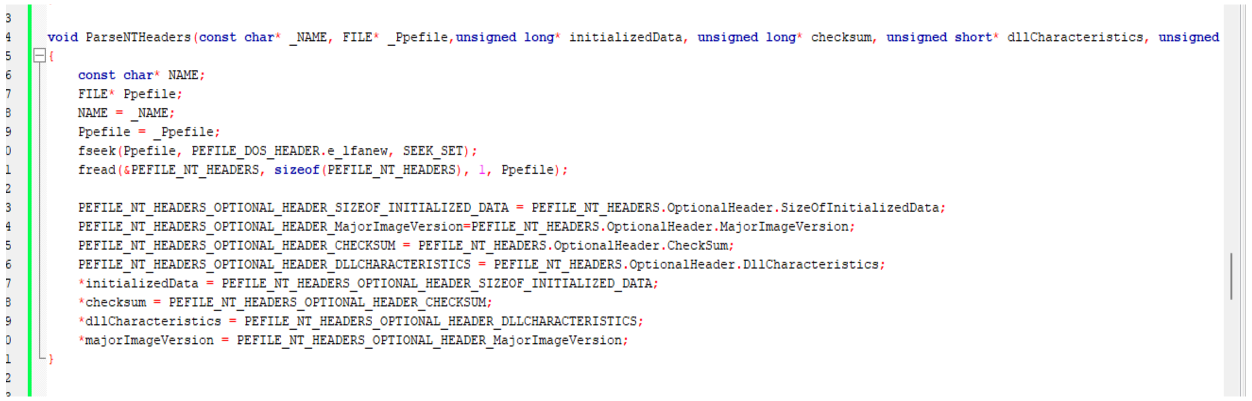
3.DLL characteristics

4.checksum

5.section names

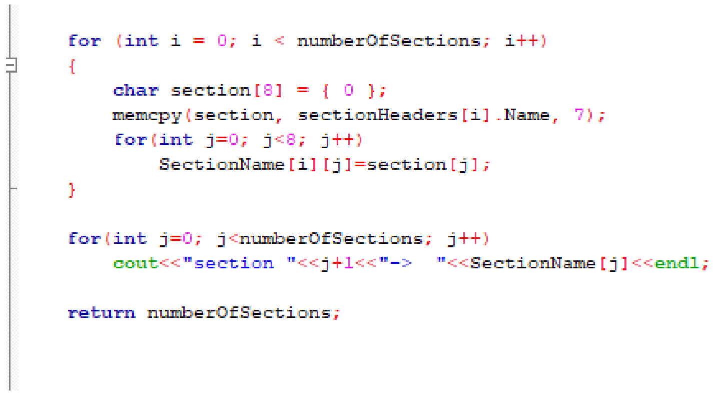
For all of the above information parsed the image DOS header, Image file header,optional header and the section header. If the evaluated values are matches with the malwares characteristics, then the file is malware.

Parsing NT header that evaluates the desired four information in the following part-



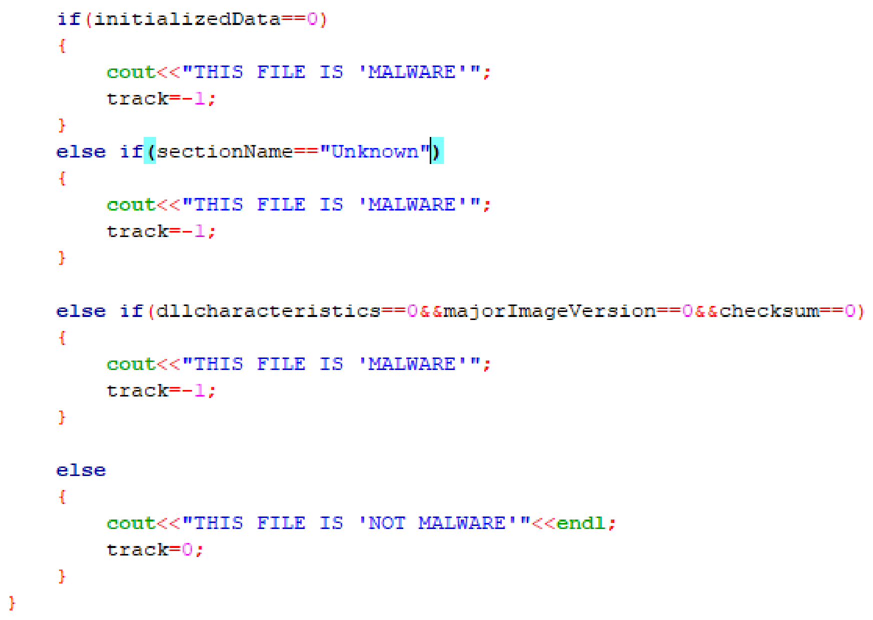
```
void ParseNTHeaders(const char* _NAME, FILE* _Ppefile,unsigned long* initializedData, unsigned long* checksum, unsigned short* dllCharacteristics, unsigned
{
    const char* NAME;
    FILE* Ppefile;
    NAME = _NAME;
    Ppefile = _Ppefile;
    fseek(Ppefile, PEFILE_DOS_HEADER.e_lfanew, SEEK_SET);
    fread(&PEFILE_NT_HEADERS, sizeof(PEFILE_NT_HEADERS), 1, Ppefile);

    PEFILE_NT_HEADERS_OPTIONAL_HEADER_SIZEOF_INITIALIZED_DATA = PEFILE_NT_HEADERS.OptionalHeader.SizeOfInitializedData;
    PEFILE_NT_HEADERS_OPTIONAL_HEADER_MajorImageVersion=PEFILE_NT_HEADERS.OptionalHeader.MajorImageVersion;
    PEFILE_NT_HEADERS_OPTIONAL_HEADER_CHECKSUM = PEFILE_NT_HEADERS.OptionalHeader.CheckSum;
    PEFILE_NT_HEADERS_OPTIONAL_HEADER_DLLCHARACTERISTICS = PEFILE_NT_HEADERS.OptionalHeader.DllCharacteristics;
    *initializedData = PEFILE_NT_HEADERS_OPTIONAL_HEADER_SIZEOF_INITIALIZED_DATA;
    *checksum = PEFILE_NT_HEADERS_OPTIONAL_HEADER_CHECKSUM;
    *dllCharacteristics = PEFILE_NT_HEADERS_OPTIONAL_HEADER_DLLCHARACTERISTICS;
    *majorImageVersion = PEFILE_NT_HEADERS_OPTIONAL_HEADER_MajorImageVersion;
}
```

Finding section names using this part-



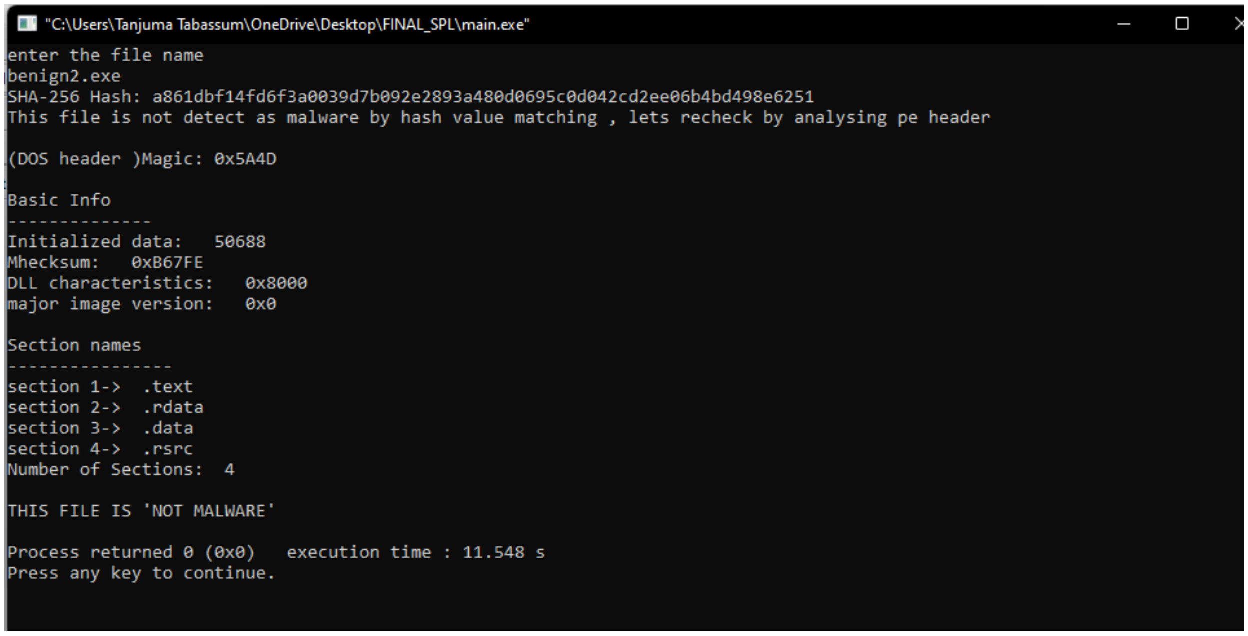
```
    for (int i = 0; i < numberOfSections; i++)
    {
        char section[8] = { 0 };
        memcpy(section, sectionHeaders[i].Name, 7);
        for(int j=0; j<8; j++)
            SectionName[i][j]=section[j];
    }

    for(int j=0; j<numberOfSections; j++)
        cout<<"section "<<j+1<<"->  "<<SectionName[j]<<endl;

    return numberOfSections;
```

For combining the evaluated informations , took a decision using the algorithm implemented here whether the file malware or not



```
        if(initializedData==0)
        {
            cout<<"THIS FILE IS 'MALWARE'";
            track=-1;
        }
        else if(sectionName=="Unknown")
        {
            cout<<"THIS FILE IS 'MALWARE'";
            track=-1;
        }

        else if(dllcharacteristics==0&&majorImageVersion==0&&checksum==0)
        {
            cout<<"THIS FILE IS 'MALWARE'";
            track=-1;
        }

        else
        {
            cout<<"THIS FILE IS 'NOT MALWARE'"<<endl;
            track=0;
        }
    }
```

Overall output-



"C:\Users\Tanjuma Tabassum\OneDrive\Desktop\FINAL_SPL\main.exe"

```
enter the file name
benign2.exe
SHA-256 Hash: a861dbf14fd6f3a0039d7b092e2893a480d0695c0d042cd2ee06b4bd498e6251
This file is not detect as malware by hash value matching , lets recheck by analysing pe header

(DOS header )Magic: 0x5A4D

Basic Info
--------------
Initialized data:   50688
Mhecksum:   0xB67FE
DLL characteristics:   0x8000
major image version:   0x0

Section names
----------------
section 1->  .text
section 2->  .rdata
section 3->  .data
section 4->  .rsrc
Number of Sections:  4

THIS FILE IS 'NOT MALWARE'

Process returned 0 (0x0)   execution time : 11.548 s
Press any key to continue.
```

4.User Interaction: Provides an intuitive user interface that presents detection results, prompts the user for actions, and allows file deletion or quarantine.if the file is detected as malware, then the system shows a menu to users that if he want to delete the file or quarantine the file as the file is harmful for the computer system or if he wants he can have the file as it is.

5.File Management: Implemented file deletion or quarantine operations based on the user's selections. Quarantine the file using bit shifting operation.

3.User manual:

For running this project in your computer you need do the followings-

1. Download the "Malware Detecting System" from the provided source that includes all code written in c++.
2. Extract the contents of the system to a desired location on your computer
3. Compile the source code using the appropriate compiler command.
4. Execute the generated executable to launch the system.

The project done the followings step by step-

1.Give the input file path as required.

2.The project will evaluate the SHA256 hash of the input file and compare it with the known malware hashes using the trie-based string matching algorithm.

3.If a match is found, the system will label the file as malware.

4.If no match is found, the system will proceed with analyzing the PE header to further determine if the file is malicious.

5.Based on the analysis results, the user will be prompted to delete or quarantine the file.

6..If the user selects deletion, the file will be permanently removed from the system.

7.If the user selects quarantine, the file will be encoded using a 1-bit shifting operation and moved to a secure quarantine folder.

8.If the user selects “nothing”, then the file will be as it was.

4. Conclusion

The "Malware Detecting System" provides an automated and efficient solution for identifying and mitigating malware threats in executable files. By utilizing techniques such as SHA256 hashing, string matching algorithms (trie), and PE header parsing, the system can effectively detect and classify potential malware files. The user-friendly interface allows users to make informed decisions on how to handle identified malware files, ensuring the security and integrity of their computer systems.

5.References:

[1] A survey on malware and malware detecting systems, Imtithal A.Saeed,Ali selamat, Ali M. A. Abuagoub, International Journal of Computer Applications(0975- 8887), April-2013

[2] PE-Header-Based Malware Study and Detection, Yibin Liao ,Department of Computer Science The University of Georgia, Athens, GA 30605.

[3] SHA-256 Cryptographic Hash Algorithm (komodoplatform.com), komodoplatform.com, last accessed on 17.01.2023