

SPL-1 Project Report, 2022

Malware Detection System

SE 305: Software Project Lab-1

Submitted by

Tanjuma Tabassum

BSSE Roll No: 1312

BSSE Session: 2020-2021

Supervised by

Moumita Asad

**Designation: Lecturer
Institute of Information Technology**



**Institute of Information Technology
University of Dhaka
[21-05-2023]**

Contents

1.Introduction:.....	2
1.1 Background Study:.....	2
1.2 Challenges.....	5
2.Project Overview:.....	6
3.User manual:	9
4. Conclusion:.....	11
References:	11

1. Introduction

The "Malware Detection 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 data structure(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 detection system.

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

The main 4 parts of the project are-

- ❖ **SHA 256:** SHA-256, which stands for secure hash algorithm 256, is a cryptographic hashing algorithm that's used for message, file, and data integrity verification. It's part of the SHA-2 family of hash functions and uses a 256-bit key to take a piece of data and convert it into a new, unrecognizable data string of a fixed length. This string of random characters and numbers, called a hash value, is also 256 bits in size. Here SHA 256 hash value of each file uniquely identify the file. Every malware file can be uniquely identified by its hash value.

The complete process of evaluating hash value can be divided into 4 segments-

1.Padding bits: You can add 64 bits of data now to make the final plaintext a multiple of 512. You can calculate these 64 bits of characters by applying the modulus to our original cleartext without the padding.

Original message +padding bits +modulus values=final data to be hashed as multiple of 512

2.Initializing the buffers: Need to initialize the default values for eight buffers to be used in the rounds. and also need to initialize 64 different keys in an array.

3.compression function: The entire message gets broken down into multiple blocks of 512 bits each. It puts each block through 64 rounds of operation, with the output of each block serving as the input for the following block. The entire process is as follows:

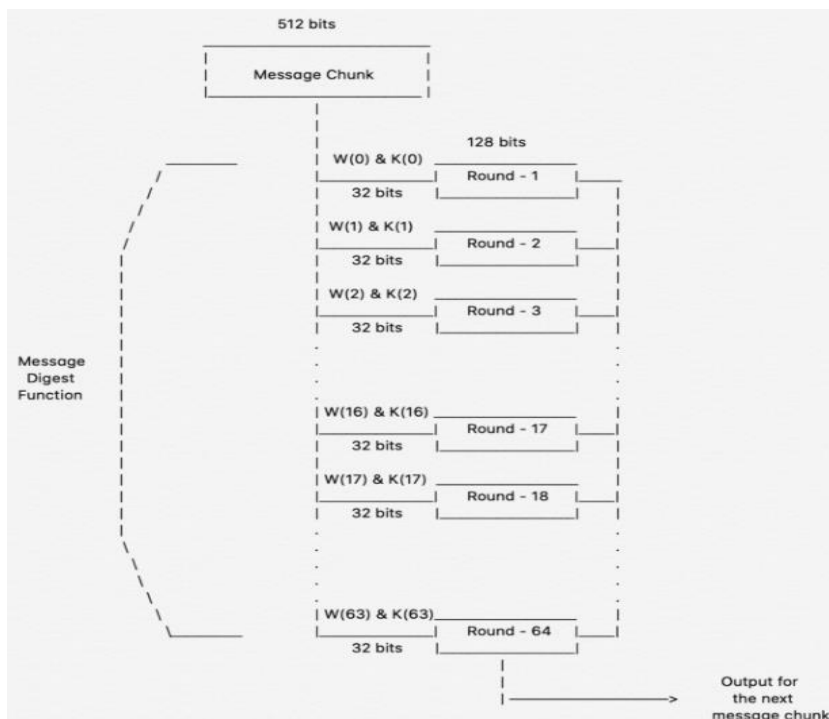


Figure- 01: SHA 256 message digest

4.Output: With each iteration, the final output of the block serves as the input for the next block. The entire cycle keeps repeating until you reach the last 512-bit block, and you then consider its output the final hash digest. This digest will be of the length 256-bit, as per the name of this algorithm. [3]

❖ **Trie data structure for string matching:** A trie is a multiway tree data structure used for storing strings over an alphabet. It is used to store a large amount of strings. The

string matching here hash matching efficiently done by tries. Firstly, I inserted all the known hash values to the trie. Then I just searched the hash value of the input file. if any match found then the file is “Malware”. A trie can be look like this-

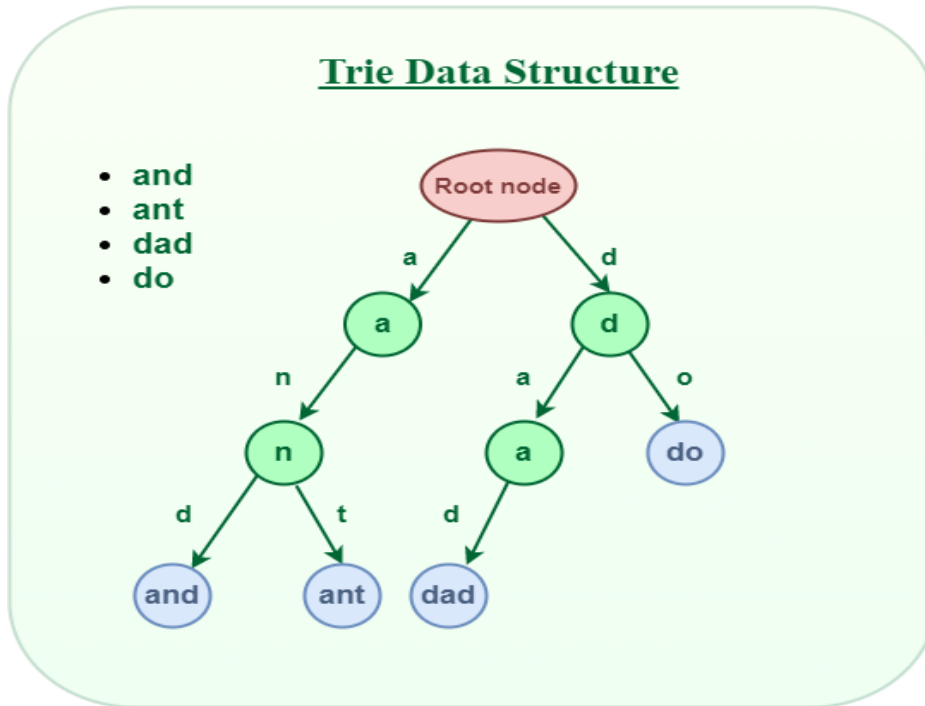


Figure- 02: Trie data structure

❖ **PE header parsing:** The portable executable file format is file format for executables, DLL, object code and others used in 32 bits, 64 bits versions of windows operating system. The PE file format is a data structure that encapsulate the necessary info for the OS loader to manage the wrapped executable code. I parse the PE header of the exe file to know the characteristics of the file if it matches to the characteristics of the malware. [2]

The format of a PE file is look like this-

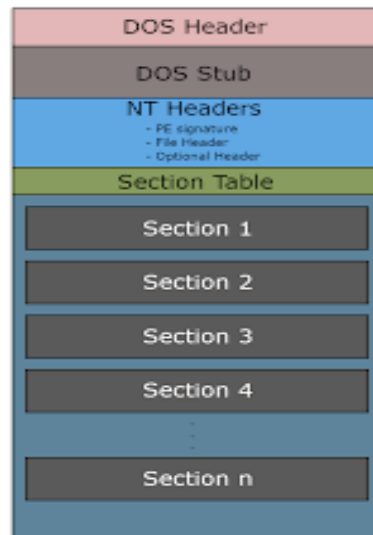


Figure-03: PE file format

- ❖ **Encoding of an exe file:** in this project I used a simple bit shifting operation to encode a malware. Encoding the file using 1 bit left shifting operation and decoding using 1 bit right shifting operation.

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 parses the necessary information of the PE file and checks the characteristics of the file to see 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 parts 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.

I Had to study all of the parts of my project because I had no idea of this project. And overcome the difficulties by analyzing the part more and more and taking help from various resources.

2. Project Overview

The project architecture consists of the following parts:

- User menu
- File/Folder 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.User menu:

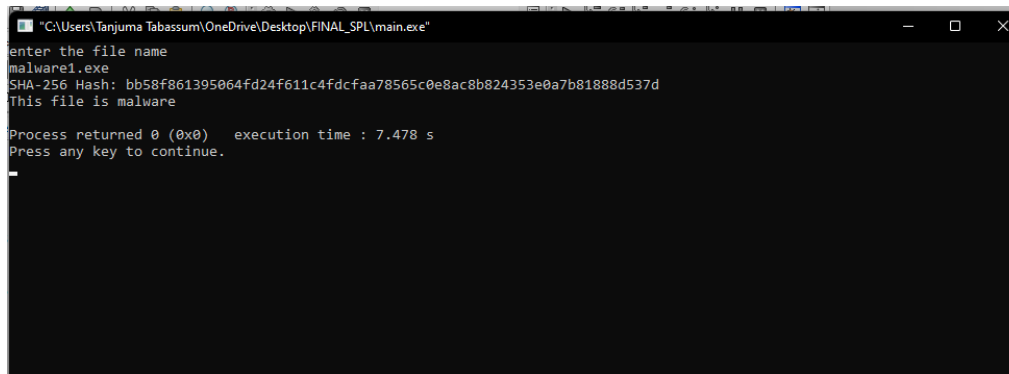
First of all, showing a menu to the user that if he wants to test a file or a folder that is malware or not.

2.SHA256 Hash value evaluation:

Calculates the SHA256 hash value of the input executable file and compares it against a file of hash values of known malwares that are collected from the site “malshare.com”[4].

3. String Matching data structure:

Utilizes a trie-based string-matching to efficiently search for matching hashes. Here I conduct two operations in trie. These are Insert and Search. I inserted all the known hash values from file to the trie. Then conduct a search operation in the inserted trie. Here’s a sample output if a file’s hash value matches with the known malware hashes-



```
"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.
```

Figure- 04: Sample output

4.PE Header Analysis:

If the input file is not detected as malware, then for further test I conduct a static analysis named “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 I parsed the image DOS header, Image file header, Image optional header and the section header. If the evaluated values match with the malware's characteristics, then the file will be malware.

For combining the evaluated information, I took a decision using the algorithm implemented here whether the file was malware or not-

```
if(initializedData==0)
{
    cout<<"THIS FILE IS 'MALWARE'";
}

else if(sectionName=="unknown")
{
    cout<<"THIS FILE IS 'MALWARE'";
}

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

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

Figure-05: Code snippet that combines the info from header parsing

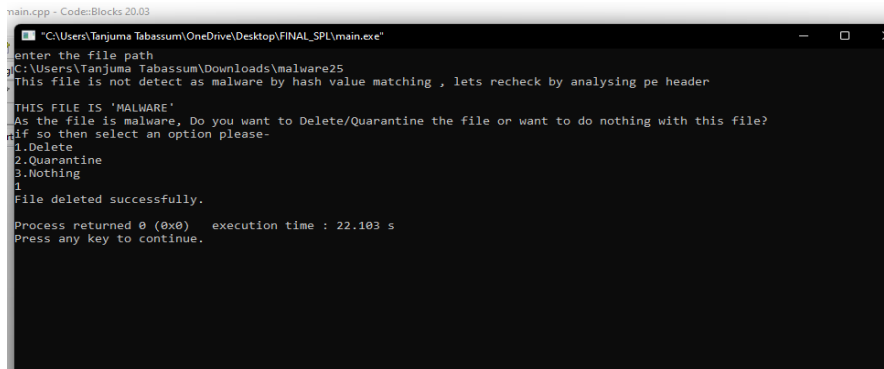
4.User Interaction:

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 the user 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 and move it to another folder named "quarantined file".

Overall output shown here when the file is detected as a malware after PE header analysis-



```
main.cpp - Code::Blocks 20.03
"C:\Users\tanjuma Tabassum\OneDrive\Desktop\FINAL_SPL\main.exe"
Enter the file path
C:\Users\tanjuma Tabassum\Downloads\malware25
This file is not detect as malware by hash value matching , lets recheck by analysing pe header
THIS FILE IS 'MALWARE'
As the file is malware, Do you want to Delete/Quarantine the file or want to do nothing with this file?
If so then select an option please-
1.Delete
2.Quarantine
3.Nothing
1
File deleted successfully.
Process returned 0 (0x0)   execution time : 22.103 s
Press any key to continue.
```

Figure-05: sample output

3. User manual

This user manual provides step-by-step instructions on how to obtain the “Malware Detection System” from GitHub. Please follow the guidelines below:

Prerequisites:

1. C++ compiler: Ensure that you have a C++ compiler installed on your system.
2. Code Editor/IDE: Have a code editor or IDE installed to open and edit the “MalwareDetection.cpp” file

Instructions:

1. Visit the SPL-1 repository on GitHub at

<https://github.com/TANJUMAJERIN/SPL-1>

2. On the GitHub repository page, click on the “Code” button and select the option “Download Zip” to download the zip file.
3. Extract the contents of the system to a desired location on your computer.

4. Compile the MalwareDetection.cpp file using C++ compiler.
5. Run the file
6. Choose an option

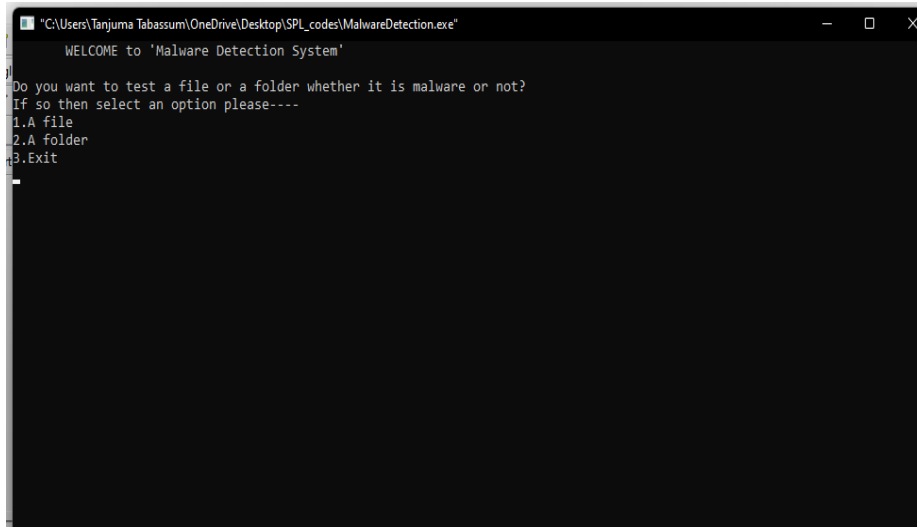


Figure-06: user menu

7. Give the path of a file or a folder

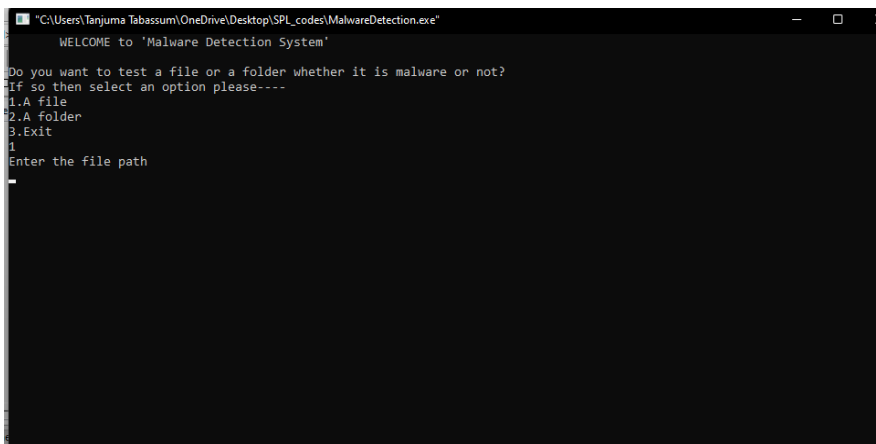
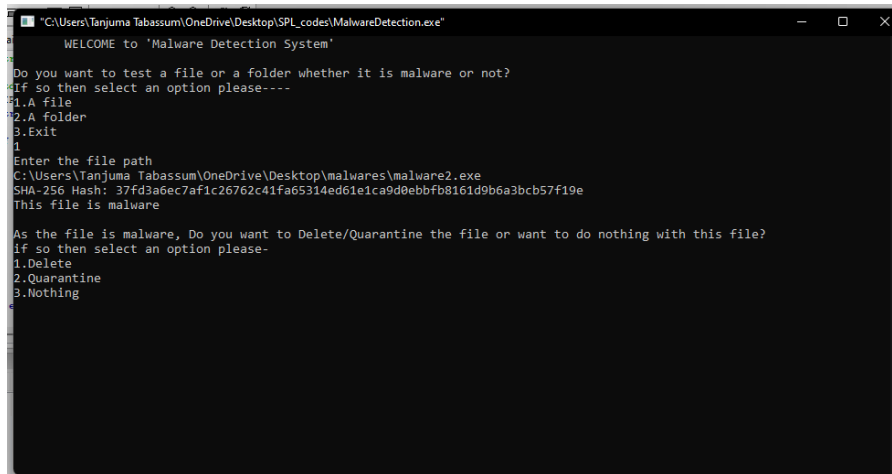


Figure-07: Take input

8. If the input file or a file from the input folder is detected as malware. Then

Choose an option



```
"C:\Users\tanjuma Tabassum\OneDrive\Desktop\SPL_codes\MalwareDetection.exe"
WELCOME to 'Malware Detection System'

Do you want to test a file or a folder whether it is malware or not?
If so then select an option please----
1.A file
2.A folder
3.Exit
1
Enter the file path
C:\Users\tanjuma Tabassum\OneDrive\Desktop\malwares\malware2.exe
SHA-256 Hash: 37fd3a6ec7af1c26762c41fa65314ed61e1ca9d0ebbf8161d9b6a3bcb57f19e
This file is malware

As the file is malware, Do you want to Delete/Quarantine the file or want to do nothing with this file?
if so then select an option please-
1.Delete
2.Quarantine
3.Nothing
1
```

Figure-08: user menu

9. The system then done operation(Deletion/quarantine) according to your choice.

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.

References

- [1] A survey on malware and malware detecting systems, Imtithal A.Saeed, Ali selamat, Ali M. A. Abu Ayoub, 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]<https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256>,algorithm simplilearn,
january-2023

[4]<https://malshare.com>