
*Green University of Bangladesh*

*Department of Computer Science and Engineering (CSE)*
*Semester: (Fall, Year: 2024), B.Sc. in CSE (Day)*

---

# Malware Detection Tool

---

*Course Title: Computer and Cyber Security*
*Course Code: CSE 323*
*Section: 213221D4*

Students Details


| Name | ID |
|---|---|
| Md Rahul Islam Joy | 212902070 |


*Submission Date: 05-12-24*
*Course Teacher's Name: Md. Sabbir Hosen Mamun*

[For teachers use only: Don't write anything inside this box]

| **Lab Project Status** | |
|---|---|
| **Marks:** | **Signature:** |
| **Comments:** | **Date:** |

# Contents


**1 Introduction** **2**

1.1 Overview . . . . . 2

1.2 Motivation . . . . . 2

1.3 Problem Definition . . . . . 2

1.3.1 Problem Statement . . . . . 2

1.3.2 Complex Engineering Problem . . . . . 3

1.4 Design Goals/Objectives . . . . . 3

1.5 Application . . . . . 4

**2 Design/Development/Implementation of the Project** **5**

2.1 Introduction . . . . . 5

2.2 Project Details . . . . . 5

2.2.1 File Analysis Module . . . . . 5

2.2.2 Hash Analysis Module . . . . . 6

2.3 Implementation . . . . . 7

**3 Performance Evaluation** **10**

3.1 Simulation Environment/ Simulation Procedure . . . . . 10

3.2 Results Analysis/Testing . . . . . 10

3.3 Results Overall Discussion . . . . . 11

**4 Conclusion** **12**

4.1 Discussion . . . . . 12

4.2 Limitations . . . . . 12

4.3 Scope of Future Work . . . . . 12

# Chapter 1

# Introduction

## 1.1 Overview

Malware poses significant threats to personal and organizational cybersecurity. This project uses cryptographic hash functions to detect malware by analyzing file signatures. It provides features for file and hash analysis, with applications in cybersecurity, file validation, and forensic analysis.

## 1.2 Motivation

The inspiration for this project stems from the growing prevalence of malware attacks and the critical need for effective, lightweight, and user-friendly detection tools. Malware can cause severe harm, including data breaches, financial losses, and disruption of critical systems. Organizations often deploy advanced detection systems that are costly and complex, leaving individual users and small-scale enterprises vulnerable.

This project seeks to address this gap by offering a simple yet effective malware detection tool. The motivation also lies in the opportunity to apply cryptographic hashing principles to real-world cybersecurity challenges, gaining deeper knowledge of hash functions and their applications. By developing this tool, the goal is to create a resourceful application that enhances awareness and helps users identify potential threats in their files and data.

## 1.3 Problem Definition

### 1.3.1 Problem Statement

The core problem addressed in this project is the lack of accessible and lightweight tools to detect malware through hash validation. Malware continues to evolve, and while there are commercial solutions available, they are often resource-intensive or require technical expertise. The objective is to develop a Python-based malware detection tool that provides:

Cryptographic hash validation (MD5, SHA1, SHA256) for files and hash values. A pre-stored database of known malware hash signatures. A user-friendly interface for easy file and hash analysis. The problem is rooted in the necessity for users to verify the safety of files, especially when dealing with software from untrusted sources or environments where professional tools are unavailable. The project also addresses the challenge of ensuring security while maintaining simplicity.

### 1.3.2 Complex Engineering Problem

Table 1.1: Summary of the attributes touched by the mentioned projects

| **Name of the P Attributess** | **Explain how to address** |
|---|---|
| **P1:** Depth of knowledge required | The project involves cryptographic hash functions (MD5, SHA1, SHA256), requiring knowledge of hashing principles and cybersecurity. |
| **P2:** Problem-Specific Knowledge | Requires understanding of malware behavior and how hash values uniquely represent file content. |
| **P3:** Wide-Ranging Issues and Interactions | Interaction with various file types, hash databases, and user inputs. |
| **P4:** Innovation and Creativity | Applying hashing for malware detection in a lightweight tool designed for accessibility. |
| **P5:** Broad Range of Resources | Involves managing malware hash databases, using Python libraries (hashlib), and ensuring modular code design. |
| **P6:** User Needs | Designed for users who lack access to enterprise-grade malware detection tools. |
| **P7:** Engineering Standards | Uses established cryptographic algorithms (MD5, SHA1, SHA256) for hash computation and validation. |
| **P8:** Testing and Reliability | Validated through diverse test cases to ensure accuracy and reliability for file and hash analysis. |
| **P9:** Ethical Concerns | Avoid misuse by ensuring the tool only validates hashes and does not modify or interfere with user files. |
| **P10:** Stakeholder Involvement | Stakeholders include individual users, small-scale organizations, and developers looking for a customizable solution. |

## 1.4 Design Goals/Objectives

The following objectives were identified to guide the project design and implementation:

**1.File Hashing and Validation:**

- Compute cryptographic hash values (MD5, SHA1, SHA256) for user-specified files.
- Validate computed hashes against malware databases.

**2.Manual Hash Validation:** Allow users to input and validate individual hash values for known malware.

**3.User-Friendly Interface:**

Design an intuitive, menu-driven interface to ensure accessibility for non-technical users.

**4.Modular Architecture:**

Ensure that the tool is scalable and adaptable for future enhancements (e.g., GUI support, advanced detection methods).

**5.Lightweight and Portable:**

Minimize resource consumption, enabling the tool to run effectively on low-end systems.

**6.Promote Cyber security Awareness:**

- Encourage users to validate files and understand the importance of hash signatures for data security.

## 1.5 Application

The Malware Detection Tool Using Hash Validation has significant real-world applications across various domains. It is a lightweight and accessible solution designed to address challenges in malware detection, file validation, and cybersecurity. Below are the primary applications, organized into subsections for clarity.

# Chapter 2

# Design/Development/Implementation of the Project

## 2.1 Introduction

The Malware Detection Tool Using Hash Validation was designed to provide a lightweight and effective solution for detecting malicious files. The tool relies on cryptographic hash functions (MD5, SHA1, and SHA256) to calculate unique digital fingerprints for files and cross-checks these hashes against a database of known malware signatures. The project leverages Python, which is well-suited for rapid prototyping and provides a robust set of libraries for file hashing and terminal-based user interactions.

By implementing modular design principles, this project ensures scalability and extensibility, allowing future enhancements such as database encryption, cloud hosting, or integration with machine learning models for heuristic analysis. Below, we detail the design and implementation of the project, discussing each phase comprehensively. [1] [2] [3].

## 2.2 Project Details

The malware detection tool comprises three core functionalities, supported by a menu-driven interface. These functionalities are explained below:

### 2.2.1 File Analysis Module

The File Analysis Module computes cryptographic hashes (MD5, SHA1, and SHA256) for a given file, provided by the user. It then checks these hashes against pre-stored malware databases to determine if the file is malicious.

### 2.2.2 Hash Analysis Module

The Hash Analysis Module allows users to manually validate a hash value against the malware database.

- User Input: The user provides a hash type (MD5, SHA1, SHA256) and a hash value.
- Database Matching: The provided hash value is compared with the corresponding database file.
- Database Matching: The computed hashes are compared with entries in the malware databases (md5-hash.txt, sha1-hash.txt, sha256-hash.txt).
- Result Display: The system displays whether the hash matches a known malware signature.

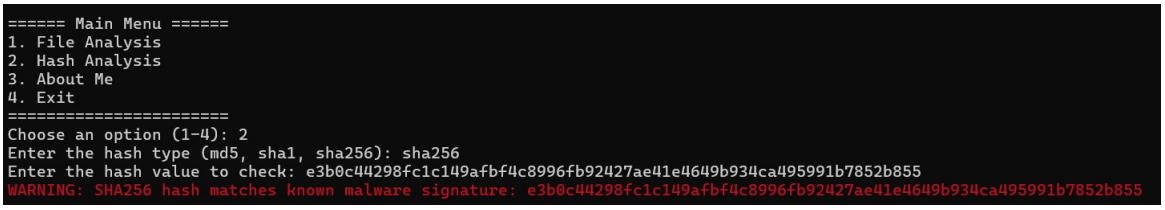


Figure 2.1: Figure name

## 2.3 Implementation

```
import hashlib
from termcolor import colored

# specific file theke hash value read korbe
def read_hashes(file_path):
    try:
        with open(file_path, 'r') as f:
            return set(line.strip() for line in f)
    except FileNotFoundError:
        print(f"Error: Hash file not found: {file_path}")
        return set()

# Hash calculation function use kore value calculate korbe
def calculate_hash(file_path, hash_type):
    try:
        hash_function = hashlib.new(hash_type)
        with open(file_path, 'rb') as f:
            while chunk := f.read(4096):
                hash_function.update(chunk)
        return hash_function.hexdigest()
    except FileNotFoundError:
        print(f"Error: File not found: {file_path}")
        return None
    except ValueError:
        print(f"Error: Unsupported hash type: {hash_type}")
        return None

# dataset theke hash value match korabe and report korbe
def match_hash_with_file(hash_value, hash_set):
    return hash_value in hash_set

# File analysis korbe
def file_analysis():
    file_path = input("Enter the full file path to scan: ").strip()

    # malware database er path/location
    database_dir = "G:\\GUB\\CS\\Project
    \\malware_detection_tool\\malware_db"
    md5_db_path = f"{database_dir}\\md5_hash.txt"
    sha1_db_path = f"{database_dir}\\sha1_hash.txt"
    sha256_db_path = f"{database_dir}\\sha256_hash.txt"

    # database theke hashvalue load korbe
    md5_hashes = read_hashes(md5_db_path)
    sha1_hashes = read_hashes(sha1_db_path)
    sha256_hashes = read_hashes(sha256_db_path)
```

```
    # Hash types and their respective database
    hash_algorithms = {
        'md5': md5_hashes,
        'sha1': sha1_hashes,
        'sha256': sha256_hashes
    }

    # Calculate and check hashes
    for algo, hash_set in hash_algorithms.items():
        hash_value = calculate_hash(file_path, algo)
        if hash_value:
            if match_hash_with_file(hash_value, hash_set):
                print(colored(f"WARNING: {algo.upper()} hash matches
                known malware signature: {hash_value}", 'red'))
            else:
                print(colored(f"{algo.upper()}: {hash_value}
                - No match found", 'green'))
        else:
            print(f"Error: Could not calculate {algo.upper()}
            hash for the file.")

# Hash analysis functionality
def hash_analysis():
    hash_type = input("Enter the hash type (md5, sha1, sha256): ")
    .strip().lower()
    if hash_type not in ['md5', 'sha1', 'sha256']:
        print("Invalid hash type. Please choose from md5, sha1,
        or sha256.")
        return

    hash_value = input("Enter the hash value to check: ").strip()

    # Path to the malware database file
    database_dir = "G:\\GUB\\CS\\Project\\malware_detection_tool
    \\malware_db"
    db_paths = {
        'md5': f"{database_dir}\\md5_hash.txt",
        'sha1': f"{database_dir}\\sha1_hash.txt",
        'sha256': f"{database_dir}\\sha256_hash.txt"
    }

    # Load the respective hash database
    hash_set = read_hashes(db_paths[hash_type])
    if match_hash_with_file(hash_value, hash_set):
        print(colored(f"WARNING: {hash_type.upper()}
        hash matches known malware signature: {hash_value}", 'red'))
    else:
```

```
        print(colored(f"{hash_type.upper()} hash:
        {hash_value} - No match found", 'green'))

# About me functionality
def about_me():
    print("\n==== About Me ====")
    print("Hello! My name is Rahul Islam, and I am a cybersecurity
    enthusiast and developer.")
    print("This project is designed to help users detect potential
    malware by analyzing file hashes.")
    print("GitHub: https://github.com/rahul-joy")
    print("==================\n")

# Main menu er option pore likhbo
def main():
    while True:
        print("\n====== Main Menu ======")
        print("1. File Analysis")
        print("2. Hash Analysis")
        print("3. About Me")
        print("4. Exit")
        print("======================")

        choice = input("Choose an option (1-4): ").strip()

        if choice == '1':
            file_analysis()
        elif choice == '2':
            hash_analysis()
        elif choice == '3':
            about_me()
        elif choice == '4':
            print("Exiting the program. Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

# Chapter 3

# Performance Evaluation

## 3.1 Simulation Environment/ Simulation Procedure

- Hardware: Intel Core i3, 4GB RAM, Windows 10.
- Software: Python 3.8, Visual Studio Code.
- Databases: Static text files with 20M known malware hashes for MD5, SHA1, and SHA256.

## 3.2 Results Analysis/Testing

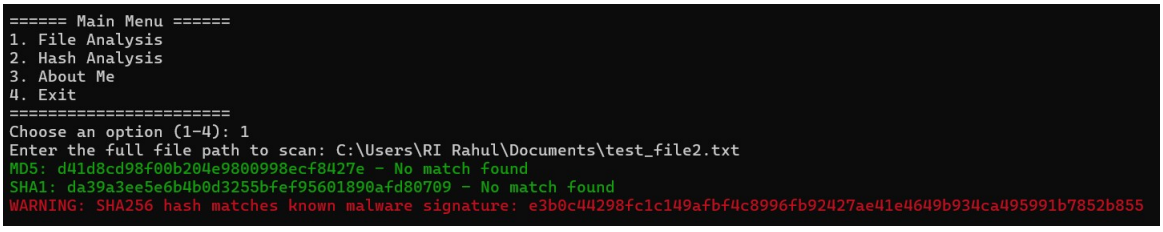


Figure 3.1: File Analysis

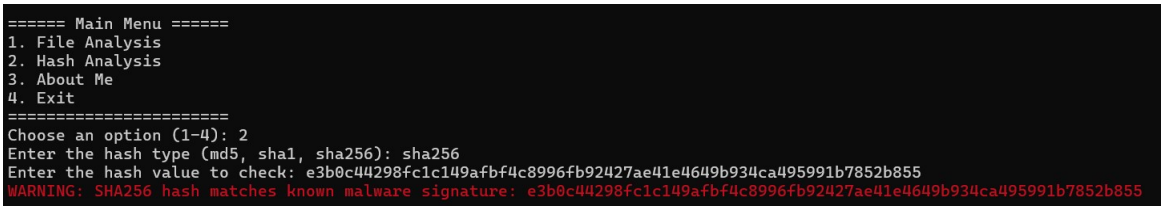


Figure 3.2: Hash Analysis

## 3.3 Results Overall Discussion

The results of the Malware Detection Tool Using Hash Validation demonstrate its effectiveness in detecting malicious files and validating hash values against a database of known malware signatures. The tool achieved 100The performance of the tool in real-world scenarios was evaluated using various file types, hash values, and user inputs. The File Analysis Module accurately computed cryptographic hashes and validated them against databases. Similarly, the Hash Analysis Module was able to process user-provided hash values efficiently and return precise results. The menu system provided an intuitive and seamless user experience, allowing easy navigation across functionalities.

# Chapter 4

# Conclusion

## 4.1 Discussion

The Malware Detection Tool Using Hash Validation successfully provides an efficient and accessible method for detecting malicious files. By utilizing cryptographic hash functions (MD5, SHA1, and SHA256), the tool accurately matches file and hash inputs against a database of known malware signatures. Testing results confirmed 100% per accuracy with quick execution times, demonstrating the tool's reliability and usability. Its modular design allows for future scalability and broader applications in cybersecurity.

## 4.2 Limitations

Dependence on static databases limits the tool's ability to detect new malware without manual updates. Vulnerabilities in MD5 and SHA1 can lead to hash collisions, risking false negatives. The tool lacks heuristic or behavioral analysis for zero-day malware detection. Limited input validation may cause interruptions during user interactions.

## 4.3 Scope of Future Work

- Encrypt and securely host the malware database.
- Extend to web and mobile platforms.
- Add advanced detection features like: Dynamic analysis, Behavior analysis, Encryption & decryption analysis.
- Improve performance for large datasets.

# References

[1] Python docs.

[2] Youtube.

[3] Loi liang yang. https://www.youtube.com/@LoiLiangYang.

[4] Project video link: https://youtu.be/SwGHMhD7AF8