Tribhuvan University

Faculty of Humanities and Social Sciences

File Compression and Malware Detection System using Decision Tree and LZMA Algorithm

A PROJECT REPORT

Submitted To

Department of Bachelor in Computer Application

Mechi Multiple Campus

In partial fulfillment of the requirements for the Bachelor in Computer Application

Submitted By

Santosh Bhandari (Roll No. 202129)

September 2024

Under the Supervision of

Raju Poudel

Tribhuvan University

Faculty of Humanities and Social Sciences

Mechi Multiple Campus

Supervisor's Recommendation

I hereby recommend that this project prepared under my supervision by SANTOSH BHANDARI entitled "**FILE COMPRESSION AND MALWARE DETECTION SYSTEM USING DECISION TREE AND LZMA ALGORITHM**" in partial fulfillment of the requirements for the degree of Bachelor of Computer Application is recommended for the final evaluation.

SIGNATURE

Raju Poudel

SUPERVISOR

Tribhuvan University

Faculty of Humanities and Social Sciences

Mechi Multiple Campus

LETTER OF APPROVAL

This is to certify that this project prepared by SANTOSH BHANDARI entitled "**FILE COMPRESSION AND MALWARE DETECTION SYSTEM USING DECISION TREE AND LZMA ALGORITHM**" in partial fulfillment of the requirements for the degree of Bachelor in Computer Application has been evaluated. In my opinion it is satisfactory in the scope and quality as a project for the required degree.

SIGNATURE of Supervisor Raju Poudel (Teaching Assistant)	**SIGNATURE OF HOD/ Coordinator**
SIGNATURE of Internal Examiner	**SIGNATURE of External Examiner**

Abstract

The File Compression and Malware Detection System is a web-based application developed using Django, designed to allow users to securely upload, scan, and compress files. The system integrates a Decision Tree algorithm to detect any malware in uploaded files, ensuring that no malicious content is present before proceeding with compression. If malware is detected, the system immediately issues a warning, halts further actions, and removes the infected file from the server.

For clean files, the system applies the Deflate algorithm, a lossless compression method that reduces file size while preserving the integrity of the data. This combination of malware detection and file compression ensures that only secure files are compressed and made available for download, making it suitable for environments where both security and storage optimization are essential. The system provides a seamless, user-friendly interface, allowing users to easily upload files, view scan results, and download compressed files securely. By merging these two functionalities into a single platform, the system effectively enhances both file security and storage efficiency, offering an optimal solution for secure file management.

Keywords:- HTML5, Django, Decision Tree, LZMA, Code Editor, JS, Model, Malware

Acknowledgment

I express my sincere gratitude for the successful completion of this project, developed as part of our BCA degree program under the expert guidance and supervision of lecturer Mr. Raju Poudel.

Supervisor Mr. Raju Poudel played an instrumental role in shaping this project from its initial conception to its execution. His extensive knowledge, insightful feedback, and continuous support were invaluable throughout this journey. We have had the privilege of learning from his expertise, which has enriched our academic experience and contributed significantly to this project's quality.

Additionally, I want to acknowledge the assistance and support from teachers, classmates and friends, who provided encouragement and valuable insights that significantly contributed to the project's development.

This project represents the culmination of efforts and support from various quarters, and I am deeply grateful to everyone who has been a part of this journey.

Santosh Bhandari

Table of Contents

List of Abbreviations

DFD	Data Flow Diagram
IDE	Integrated Development Environment
LZMA	Lempel-Ziv-Markov Chain Algorithm
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
JS	JavaScript

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Introduction

In today's digital era, the rapid exchange of files over networks has become an essential part of daily activities for both individuals and businesses. However, with the increasing volume of data transfer comes a heightened risk of cyber threats, especially in the form of malware embedded within files. Malware, which can include viruses, ransomware, and trojans, poses serious risks to both individuals and organizations by compromising sensitive data and causing system disruptions. This makes it critical to ensure that files are transmitted and stored securely, while also addressing the need for efficient storage through compression.

The File Compression and Malware Detection System, developed using Django, is designed to meet these critical requirements. It provides users with a simple and intuitive interface to upload files, scan them for malware, and compress them securely. By utilizing a Decision Tree algorithm, the system can accurately detect and identify any malicious activities within files before proceeding with compression. For files that are verified as safe, the LZMA algorithm is used to compress the data, preserving file integrity while reducing storage size. This dual-functional system provides an efficient solution for secure data management, making it a valuable tool in today's digital landscape where security and storage efficiency are of utmost importance.

1.2 Problem of Statement

In today's digital era, individuals and organizations generate and handle vast amounts of data daily. Efficient data management practices, such as file compression, are essential for conserving storage space and enhancing data transmission speeds.
The Problem of the Statement are as follows:

- Efficient data management, such as file compression, is essential for conserving storage space and optimizing data transmission.
- People don’t have strong knowledge to identify Malware.
- Difficulty in maintaining accurate and up-to-date malware records.

1.3 Objectives

The Objectives are as follows:

- To Develop a System to compress files efficiently using LZMA Algorithm.
- To Develop Machine Learning based Malware Detection system that scans uploaded files for Malicious Activities before compression using Decision Tree Algorithm.

1.4 Scope and Limitation

The File Compression and Malware Detection System will allow the users to identify where the files are clean or not.

1. The system allows users to securely upload files, scan them for malware, and compress clean files.
2. Utilizes a Decision Tree algorithm for detecting malware in uploaded files.
3. Compresses clean files using the LZMA algorithm, a lossless compression method that reduces file size.
4. Issues real-time alerts when malware is detected and automatically deletes infected files from the server.
5. Ensures file integrity during compression, maintaining the original content without loss.

The Limitations of the Malware Detection System are as follows:

1. The system relies on the accuracy of the Decision Tree algorithm, which may not detect all types of malwares, particularly advanced or evolving threats.
2. Only supports file compression using the LZMA algorithm, limiting the choice of compression methods.
3. May not handle very large files efficiently, as both the scanning and compression processes could become resource intensive.
4. Requires periodic updates to the malware database to ensure the system's malware detection capabilities remain effective.

1.5 Report Organization

The Report of File Compression and Malware Deection System is organized as follows:

Chapter 1: Introduction

This chapter provides an overview of the project, including a clear problem statement. It outlines the project's objectives, scope, and limitations, setting the stage for the report.

Chapter 2: Background Study and Literature Review

In this chapter, we dive into the project's background, comprehensively understanding its functions and components. A literature review is conducted to explore previous research related to the project, including an analysis of similar systems for evaluation.

Chapter 3: System Analysis and Design

It focuses on the systematic analysis and design of the project, illustrated with charts and figures. We delve into functional requirements using use cases and other methodologies. Additionally, this chapter presents the database schema, interface design, and DFD.

Chapter 4: Implementation and Testing

This chapter details the tools and techniques utilized for project implementation. We also cover creating test cases to assess the system as individual units and as a whole.

Chapter 5: Conclusions and Future Recommendations

It provides information about the experiences and insights gained throughout the project, offering recommendations for future projects. It concludes the report by summarizing key findings and takeaways.

Chapter 6: Appendices

Chapter 6 contains supplementary materials, including screenshots and source codes for various functions and pages of the project.

Chapter 7: References

The last chapter references the sources from which content was drawn and used within this report.

Chapter 2: Background Study and Literature Review

2.1 Background Study

The rapid expansion of the internet and digital technologies has led to an unexpected rise in the exchange of data and files across networks. While this has greatly improved communication and collaboration, it has also opened the door to numerous cybersecurity threats. Malware—ranging from viruses, worms, trojans, to ransomware—has become a significant concern for individuals and organizations alike. Malicious actors often embed malware into seemingly harmless files, which, once downloaded or shared, can cause severe damage, including data breaches, system failures, and financial loss.

However, the availability of systems that combine malware detection and file compression remains limited. Users often have to rely on separate tools to scan for malware and compress files, which can lead to inefficiencies and increased security risks. This project, File Compression and Malware Detection System, aims to address this gap by creating an integrated platform that ensures secure file handling through malware scanning and compression. By integrating both the Decision Tree algorithm for malware detection and the LZMA algorithm for file compression, the system provides a comprehensive solution for secure and efficient file management in a digital environment. [1]

2.2 Literature Review

VirusTotal is one of the most popular online platforms for analyzing files and URLs for potential malware, utilizing a variety of antivirus engines and scanners. It offers a comprehensive service by aggregating results from numerous malware detection tools, which helps ensure a thorough analysis of submitted files. The service is updated continuously to include the latest malware signatures, making it highly reliable. VirusTotal also provides API access for integration into other applications, making it versatile for developers. However, one limitation is that files submitted are shared with antivirus vendors, which could raise privacy concerns. Additionally, the free version restricts users to a limited number of scans, making it less feasible for large-scale operations. [2]

Hybrid Analysis is another effective malware analysis platform that performs both static and dynamic analysis of files. It offers users comprehensive reports detailing the behavior of suspicious files and programs, allowing for a deep understanding of potential threats. The platform also supports the use of custom YARA rules, enabling users to tailor malware detection to specific needs. However, the system has a file size restriction in its free version, which limits its usefulness for larger files. Additionally, it sometimes produces false positives, which can lead to unnecessary concerns or actions. [3]

Malwarebytes is a widely recognized tool for detecting and removing malware, spyware, and ransomware. It offers real-time scanning in its premium version, which helps to identify and neutralize threats before they become problematic. The free version, however, focuses on post-infection cleanup rather than real-time protection. While Malwarebytes is efficient and easy to use, one drawback is that the real-time protection feature can occasionally slow down system performance, especially on older or less powerful devices. [4]

7-Zip is a widely used open-source file compression tool that supports several compression formats, including ZIP, RAR, and its native 7z format. Its primary advantage lies in its use of the LZMA (Lempel–Ziv–Markov chain algorithm), which provides superior compression ratios while maintaining data integrity. 7-Zip also supports AES-256 encryption for added security during compression, making it a versatile tool for both file compression and encryption. However, while 7-Zip is highly efficient in terms of compression ratio, its compression speed may lag behind other algorithms like LZMA, especially when dealing with larger files. [5]

Chapter 3: System Analysis and Design

3.1 System Analysis

3.1.1 Requirement Analysis

The requirements of the system are determined as given below.

i. **Functional Requirements**

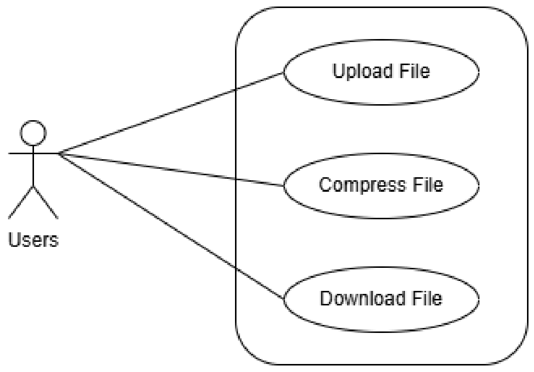


Figure 3.1.1: Use Case Diagram for File Compression & Malware Detection

Upload File: User can Upload the file for the Compression.

Compress File: If the File is clean, the User can compress the file.

Download File: The User can download the compressed file.

ii. **Non Functional Requirements**

Non-functional requirements define system attributes such as security, reliability, performance, maintainability, scalability, and usability. The non-functional requirements of the File Compression & Malware Detection are:

- **Reliability:** This System is easy to use and doesn't have high system requirements. So, the system is reliable.
- **Compatibility:** The System should work seamlessly across different web browsers and devices to provide a consistent experience.
- **Security:** The System will be secure, and the file contained the Malware is removed immediately.
- **Usability:** The system user interface is intuitive and user-friendly, ensuring users can navigate it easily.
- **Maintainability:** The System has fewer hardware requirements and can be maintained within an affordable budget in case of technical errors.

- **Response Time:** The System consistently provides timely responses to user actions to keep their experience smooth.

3.1.2 Feasibility Analysis

Before proceeding with the delopment of the system, a feasibility study is conducted to assess the viability and potential success of the project. This study evaluates various aspects, including technical feasibility, economic feasibility, and operational feasibility, to determine if the project is worth pursuing. [6]

i. Technical

Technical feasibility assesses whether the development and implementation of the proposed system are achievable using the available technology and resources. It involves evaluating the compatibility of the software solution with the existing infrastructure, hardware, and software systems.

Table 3.1.1 : Technical Feasibility

Parameters	Remarks
Hardware Resources	Available
Malware DataSet	Available
Programming Language	Free
Software Tools	Open Source / Free

The technical feasibility assessment indicates the necessary technical resources and infrastructure are available.

ii. Operational

Operational feasibility assesses whether the proposed system can be smoothly integrated into the existing operations and processes. It evaluates the system's compatibility with workflow and users' acceptance.

Table 3.1.2: Operational Feasibility

Parameters	Remarks
User Acceptance	Yes
Ease of Use	Yes
Training Requirement	Yes
Operational Cost	Affordable

The project team estimated that the remaining and running system after the deployment would not be a problem, indicating that the project is operationally feasible.

iii. Economic

Economic feasibility assesses the financial viability and benefits of developing and implementing the proposed system. It involves analyzing the costs associated with the project and comparing them with the potential benefits and returns on investment.

Table 3.1.3 : Economic Feasibility

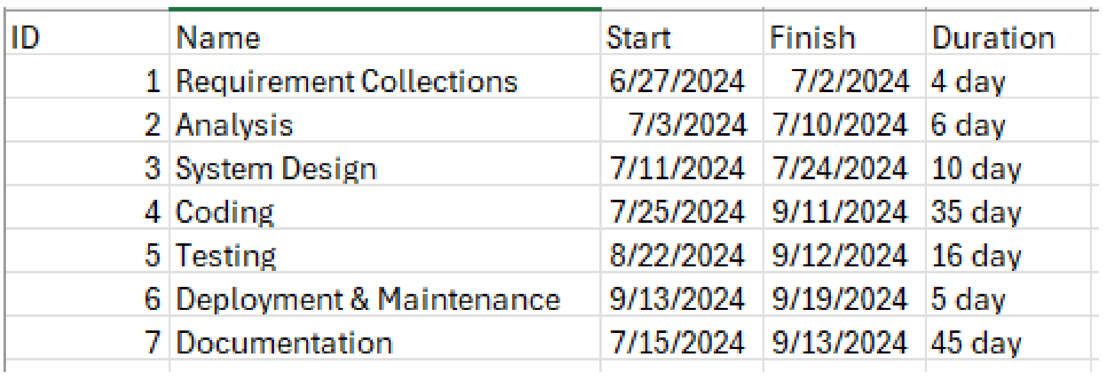
Parameters	Remarks
Development Tools	Open Source / Free
Malware DataSets	Free
IDE	Open Source / Free
Programming Language	Free

The economic feasibility study demonstrates that the proposed system can be developed cost-effectively.

iv. Schedule

In project feasibility analysis, schedule feasibility is a critical dimension to evaluate. It assesses whether the proposed project can be executed within the stipulated time frame, considering various factors that might impact project schedules.

Table 3.1.4: Schedule Feasibility



ID	Name	Start	Finish	Duration
1	Requirement Collections	6/27/2024	7/2/2024	4 day
2	Analysis	7/3/2024	7/10/2024	6 day
3	System Design	7/11/2024	7/24/2024	10 day
4	Coding	7/25/2024	9/11/2024	35 day
5	Testing	8/22/2024	9/12/2024	16 day
6	Deployment & Maintenance	9/13/2024	9/19/2024	5 day
7	Documentation	7/15/2024	9/13/2024	45 day

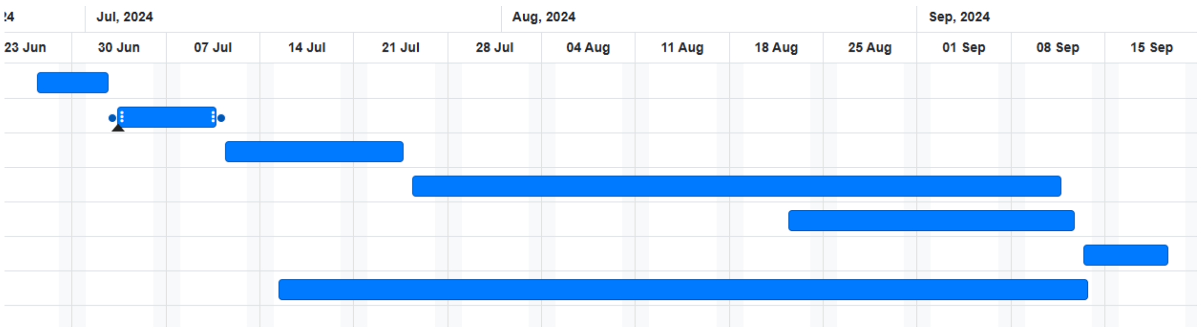


Figure 3.1.2: Gantt Chart of File Compresion & Malware Detection

3.1.3 Process Modeling(DFD)

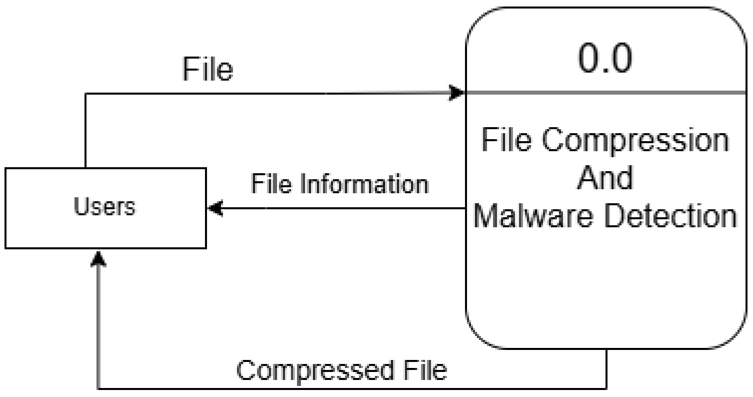


Figure 3.1.3: Context Diagram of File Compresion & Malware Detection

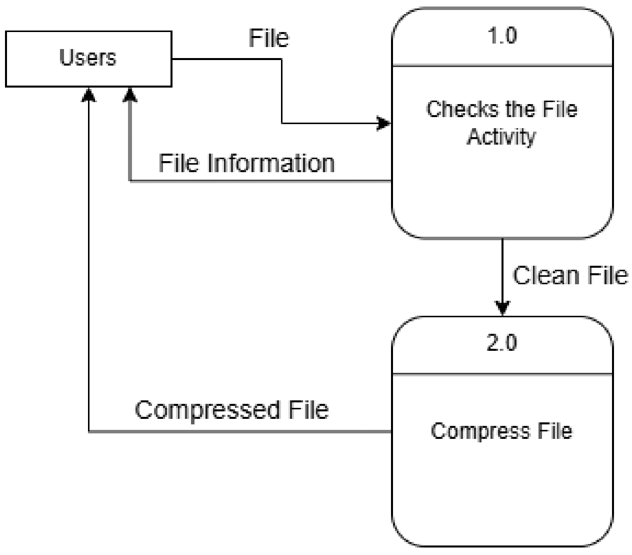


Figure 3.1.4: Level 1 DFD of File Compression & Malware Detection

3.2 System Design

3.2.1 Archietctural Design

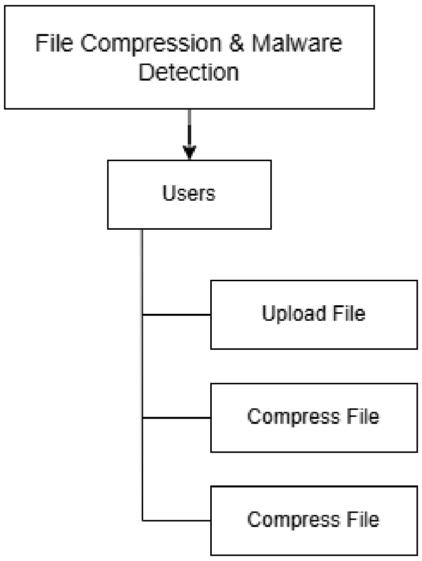


Figure 3.2.1: Architectural Design of File Compression & Malware Detection

3.2.2 Interface Design

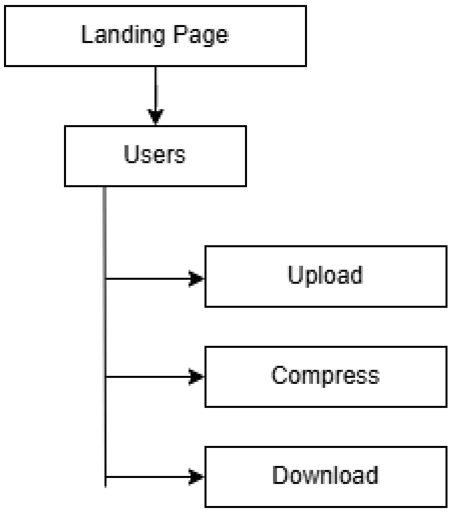


Figure 3.2.2: Interface Design of File Compression & Malware Detection

3.3Algorithm Used

This system is developed by integrating two main algorithms for its core functionalities: The Decision Tree Algorithm for malware detection and the LZMA Algorithm for file compression.

a) Decision Tree

Introduction

The Decision Tree Algorithm is a supervised learning technique that classifies files based on various features and file attributes to detect the presence of malware. It builds a tree-like structure where each node represents a decision based on the attributes, and the leaves represent the final classification outcome (malicious or clean). [7]

Where it is used?

In the File Compression and Malware Detection System, the Decision Tree algorithm is applied specifically during the malware detection phase. When a user uploads a file, the system extracts various static features from the file, such as file size, file type, entropy, and header information. These features are then passed through the trained Decision Tree model, which classifies the file as either "malicious" or "clean." The outcome of this classification determines the next step: if the file is deemed clean, it moves on to the compression process; if malware is detected, the system halts further actions and removes the file from the server.

Algorithm

i. **Feature Selection:** Extract static features from the uploaded file such as file size, file type, checksum, entropy, and header information.

ii. **Model Training:** A dataset of both malicious and clean files is used to train the Decision Tree model. The dataset consists of labeled examples that help the model learn the decision rules for classifying files.

iii. **Tree Construction:** The model builds a decision tree based on these features by iteratively splitting the dataset into subsets using the most significant feature at each node (e.g., entropy threshold or file header anomaly).

iv. **Prediction:** For each new uploaded file, the system checks the file’s attributes. The Decision Tree navigates through the decision nodes to classify the file as either "clean" or "malicious."

v. **Action:** If the file is classified as malicious, the system immediately removes the file and issues a warning to the user. If the file is clean, it proceeds to the compression stage.

b) LZMA Algorithm

Introduction

The LZMA (Lempel-Ziv-Markov chain algorithm) is used for file compression in this system. LZMA is a lossless data compression algorithm that is designed to provide a high compression ratio, making it highly efficient for reducing file sizes. It was originally developed as part of the 7z compression utility and has since been widely adopted for various applications due to its ability to compress large files into smaller sizes without losing any data integrity. [8]

Where it is used?

The LZMA algorithm is used in the File Compression and Malware Detection System to compress files once they have been verified to be free from malware. After a user uploads a file, the system scans it for potential malicious content using the Decision Tree algorithm. If the file is determined to be clean, the LZMA algorithm is applied to reduce the file size, optimizing it for storage and transmission. LZMA is known for its high compression ratio, making it ideal for scenarios where efficient storage management is required. The compressed files are saved in the .xz format, which is the default container for LZMA-compressed data. This ensures that the files maintain their original integrity while taking up significantly less space, making it suitable for environments where large amounts of data are transferred or stored. The use of LZMA helps balance storage efficiency and security in the system, providing a reliable method for handling compressed files.

Algorithm

i. **Input Preparation:** The system receives the clean file that has been validated and is ready for compression.

ii. **Compression Settings:** The LZMA algorithm is initialized with specified settings, including the compression level (preset), which defines the trade-off between compression speed and efficiency. In this implementation, a maximum compression level is set using preset 9 with the PRESET_EXTREME option.

iii. **Data Reading:** The clean file is opened for reading in binary mode. A buffer is prepared to hold the data chunks as they are read from the file.

iv. **Compression Process:** The data is processed in chunks using the LZMA compression algorithm. The algorithm identifies repeated patterns and sequences in the data, encoding them in a manner that reduces the overall file size.

v. **Output Generation:** The compressed data is written to a new file with a .xz extension, ensuring that it retains its compressed format. The system checks for successful completion of the write operation.

vi. **Completion Notification:** Once the compression is completed, the system returns the name of the compressed file and may also log the compression details for future reference.

Chapter 4: Implementation and Testing

4.1 Implementation

4.1.1 Tools Used

The following tools are used to accomplish this project:

Front End Tools

- **HTML 5**

HTML is the backbone of web pages, providing the structure and content. It uses tags to define elements such as headings, paragraphs, links, and images, allowing browsers to render web content correctly.

- **CSS 3**

CSS complements HTML by handling presentation and layout. With CSS, developers can control the visual aspects of web pages, including colors, fonts, spacing, and responsiveness. It enables the creation of visually appealing and consistent designs across different devices.

- **JS**

JavaScript is a dynamic scripting language that adds interactivity to web pages. It enables features like form validation, animations, and real-time updates without requiring a page reload. JavaScript is essential for creating engaging and user-friendly web applications.

Back End Tools

- **Django**

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. For this project, Django serves as the backbone of the system, handling user authentication, file upload, database interactions, and routing between the different functionalities of the application. Its built-in security features like protection against SQL injection and XSS attacks make it a reliable choice for developing a web-based system that handles potentially malicious files. [9]

Algorithm

- **Decision Tree Algorithm**

The Decision Tree algorithm is used for detecting malware in the uploaded files. This supervised learning algorithm builds a model that predicts the presence of malware based on the features of the file. The decision nodes in the tree are based on different characteristics of the file, such as file type, size, and content structure, allowing the system to classify whether the file is malicious or not.

- **LZMA Algorithm**

The LZMA is used for file compression in this system. LZMA is a lossless data compression algorithm that is designed to provide a high compression ratio, making it highly efficient for reducing file sizes. It was originally developed as part of the 7z compression utility and has since been widely adopted for various applications due to its ability to compress large files into smaller sizes without losing any data integrity.

Code Editor

- **Visual Studio Code**

VS Code is used to write code for this system. It is a free and lightweight source code editor supporting multiple programming languages.

Browser

- **Mozilla Firefox**

Mozilla Firefox is used to run the project. It is an open-source web browser that provides built-in tools for developers.

1.1.2 Implementation Details of Modules

According to the Needs, this system is divided into three main functional modules, i.e., the File Upload Module, Malware Detection Module and File Compression Module.

File Upload Module: This module is responsible for handling file uploads for further processing.

Malware Detection Module: The Malware Detection Module is the core of the system, responsible for scanning the uploaded files to detect potential malware. It uses the Decision Tree Algorithm to classify whether the uploaded file contains malicious content based on pre-defined file characteristics.

File Compression Module: This Module handles the compression of clean files using the Deflate Algorithm, a lossless compression technique. Once a file passes the Malware Detection scan, this module compresses the file to reduce its size for more efficient storage or transmission.

4.2 Testing

Testing is the process of executing a program to find an error. A good test case has a high probability of finding an undiscovered error. The primary objective for test case design is to derive a set of tests with the highest livelihood for uncovering defects in software. To meet this objective, two different categories of test case design are used.

4.2.1 Test Cases for Unit Testing

Table 4.2.1: Test Case for File Upload

S.N.	Uploaded File	Expected Output	Actual Output	Result
1.	Uploaded Clean File	Redirect to Compress Page	Redirect to Compress Page	Pass
2.	Uploaded Malware File	Show Warning	Showed file Contains suspicious activity.	Pass
3.	Empty Upload	Show Alert Dialog	Showd Please Select a File to Upload Alert	Pass

Table 4.2.2: Test Case for File Compression

S.N.	Input	Expected Output	Actual Output	Result
1.	Click on Compress Button	Redirect to Download Page	Compress file and redirect to Download page	Pass

Table 4.2.3: Test Case for Download File

S.N.	Input	Expected Output	Actual Output	Result
1.	Click on Download Button	Download the File	Started to Download a File	Pass

4.2.2 Test Cases for System Testing

Table 4.2.4: Test Cases For System Testing

S.N.	**Input Data**	**Expected Output**	**Actual Output**	**Result**
1.	Enter URL	Open Application Home Page	Applicaton Home Page Opended	Pass
2.	Upload a Clean file	Show Compress Page	Redirect to Compress page	Pass
3.	Upload Malware	Show Warning	Showed file Contains suspicious activity.	Pass
4.	Enter Compress URL	Show 404 Page	Redirect to 404 Page	Pass
5.	Enter Download URL	Show 404 Page	Redirect to 404 Page	Pass
6.	Enter Compress URL with valid Parameter	Show Compress Page	Showed Compress Page	Pass
7.	Enter Download URL with valid Parameter	Show Download Page	Showed Download Page	Pass
8.	Enter Random URL	Show 404 Page	Redirect to 404 Page	Pass

Chapter 5: Conclusion and Future Recommendations

5.1 Lesson Learnt / Outcome

The development of the File Compression and Malware Detection System provided valuable insights into the integration of cybersecurity measures and data management techniques. One key lesson learned was the importance of employing robust algorithms, such as Decision Tree for malware detection and LZMA for file compression, to enhance both security and efficiency. The project highlighted the need for a user-friendly interface that simplifies complex processes, making secure file handling accessible to a broader audience. Additionally, the challenges faced during implementation, including optimizing performance and ensuring data integrity, underscored the significance of thorough testing and iterative development.

5.2 Conclusion

The File Compression and Malware Detection System successfully addresses the dual challenges of ensuring file security and optimizing storage efficiency in a digital environment. By leveraging advanced algorithms such as Decision Tree for malware detection and LZMA for compression, the system provides a comprehensive solution that not only identifies and mitigates potential threats but also significantly reduces file sizes for easier management and transmission.

5.3Future Recommendations

For the improvement of the system, the following things can be done:

i. **Enhanced Malware Detection Techniques**: Integrate more advanced machine learning algorithms, such as neural networks or ensemble methods, to improve the accuracy of malware detection.

ii. **Cloud Integration**: Consider incorporating cloud storage options for uploaded files and compressed outputs, allowing users to access their files from anywhere while maintaining security and efficiency.

iii. **Real-Time Scanning**: Implement real-time scanning capabilities to monitor file uploads continuously, providing immediate feedback to users. This could enhance user experience and security by identifying threats before they are fully processed.

These recommendations aim to enhance the functionality and user experience of the system while adapting to the evolving needs of the present world. [10]

Appendices

Screen Shots

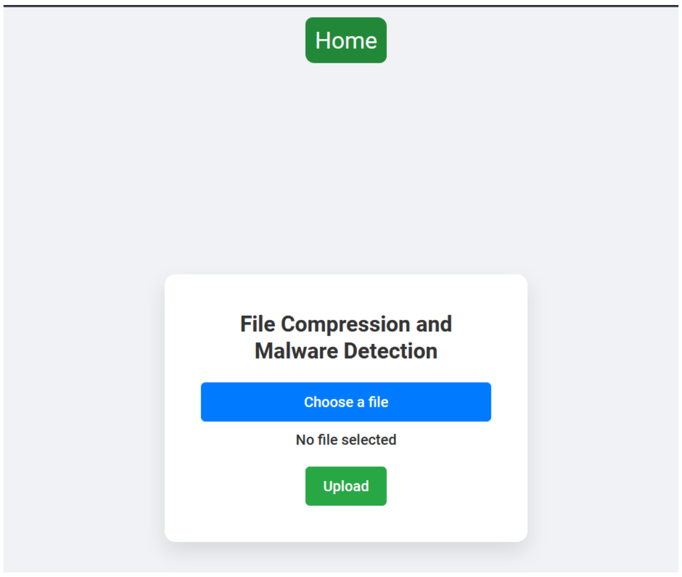
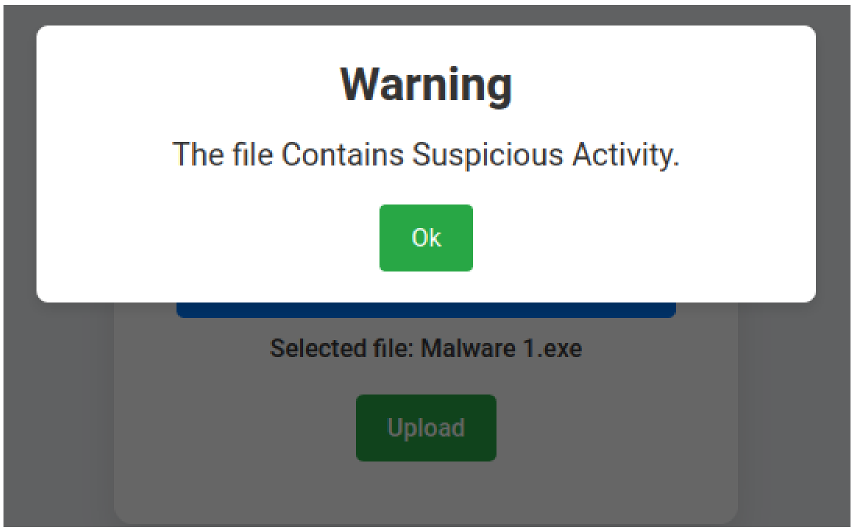


Figure 3.1 : Home Page


Warning

The file Contains Suspicious Activity.

Selected file: Malware 1.exe

Figure 6.2: Malware Detected Page

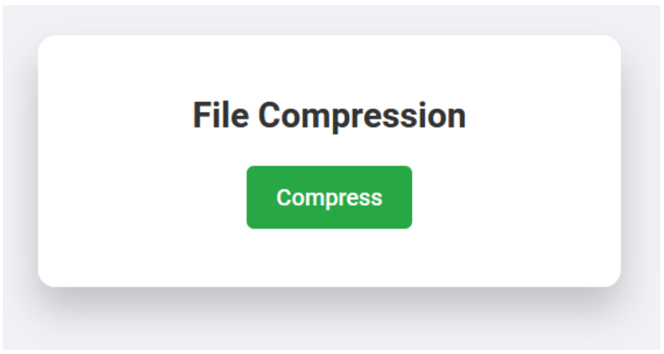


Figure 6.3: Compress Page

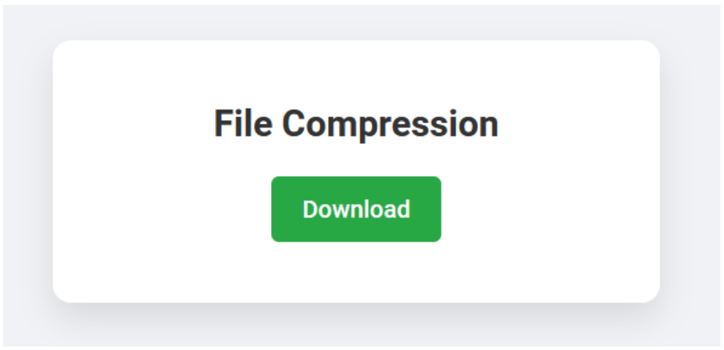


Figure 6.4: Download Page

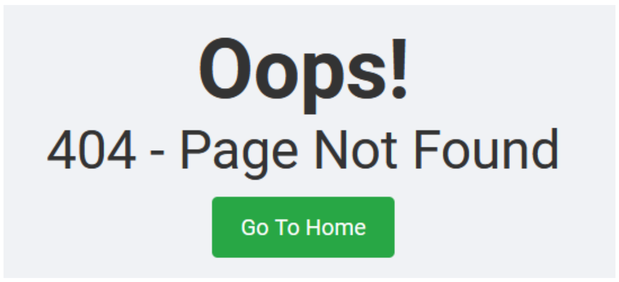


Figure 6.5: 404 Page

Source Codes

urls.py

```
from django.urls import path
from Main import views
urlpatterns = [
    path('', views.Home, name='Home'),
    path('404/',views.Custom404,name='Custom404'),
    path('compress/',views.Compression,name="Compress"),
    path("download/",views.Download,name="Download"),
    path("<slug>/",views.CustomError,name="CustomError"),
]
```

views.py

```
import pickle
from django.shortcuts import render,redirect,HttpResponse
from django.http import JsonResponse
from django.core.files.storage import FileSystemStorage
import os
from core.settings import BASE_DIR,MEDIA_ROOT
from Main.FeaturesExtraction import extract_features
from Main.fileCompress import CompressFile
# Load the trained model
model_path = os.path.join(BASE_DIR, 'MalwareModel/Malware.pkl')
with open(model_path, 'rb') as model_file:
    model = pickle.load(model_file)
# Home View
def Home(request):
    if request.method == 'POST' and request.FILES['file']:
        uploaded_file = request.FILES['file']
        fs = FileSystemStorage(location=os.path.join(MEDIA_ROOT,
'UploadFiles'))
        file_path_temp = fs.save(uploaded_file.name, uploaded_file)
        file_path = os.path.join(BASE_DIR, 'media/UploadFiles',
file_path_temp)
```

```
        if not os.path.exists(file_path):
            return JsonResponse({'message': 'File not found.', 'malware': False})
        file_features = extract_features(file_path)
        is_malware = model.predict([file_features])[0]
        md5hash=calculateMD5Hash(file_path)
        if is_malware == 1:
            os.remove(file_path)
                return JsonResponse({'message':'The file Contains Suspicious
Activity.','malware': True})
        else:
            return JsonResponse({'message': 'The file is safe and has been
compressed.', 'malware': False, 'fileName': file_path_temp})
    return render(request, 'index.html')
# File Compression View
def Compression(request):
    if request.method == 'GET':
        filename = request.GET.get('filename')
        if not filename:
            return redirect("/404")
        filepath= os.path.join(BASE_DIR, 'media/UploadFiles', filename)
        if not os.path.exists(filepath):
            return redirect("/404")
        return render(request,'compress.html',{'filename':filename})

    if request.method=="POST" and request.POST['filename']:
        filename=request.POST.get('filename')
        if not filename:
            return redirect("/404")
        filepath= os.path.join(BASE_DIR, 'media/UploadFiles', filename)
        if not os.path.exists(filepath):
            return redirect("/404")
        compressedfilename=CompressFile(filename)
        if(compressedfilename):
            return JsonResponse({'message': 'File compressed successfully.',
'compress': True, 'fileName': compressedfilename})
        else:
            return JsonResponse({'message': 'An error occurred while
compressing the file.', 'compress': False})
    return redirect("/404")
# File Download View
def Download(request):
    if request.method=="GET" and request.GET.get('filename'):
        if not request.GET.get('filename'):
            return redirect("/404")
        if not os.path.exists(os.path.join(BASE_DIR,
'media/CompressedFiles/', f"{request.GET.get('filename')}")):
            return redirect("/404")
        print("File Name : ",request.GET.get('filename'))
        return
render(request,'download.html',{'filename':request.GET.get('filename')})
    if request.method=="POST":
        filename=request.POST.get('filename')
        print(filename)
        if not filename:
            return redirect("/404")
        filepath= os.path.join(BASE_DIR, 'media/CompressedFiles/',
f"{filename}")
        if not os.path.exists(filepath):
            print("file not found")
            return redirect("/404")
        with open(filepath, 'rb') as file:
            response = HttpResponse(file.read(), content_type="application/x-
xz")
            response['Content-Disposition'] = f'attachment;
filename={filename}'
            return response
    return redirect("/404")
```

```
# Custom 404 Error View
def Custom404(request):
    return render(request,"404.html")
# Custom Error View
def CustomError(request,slug):
    return redirect("/404")
```

FeaturesExtraction.py

```
import pefile
import numpy as np
def extract_features(file_path):
    try:
        pe = pefile.PE(file_path)
        features = [
            pe.FILE_HEADER.Machine,
            pe.FILE_HEADER.SizeOfOptionalHeader,
            pe.FILE_HEADER.Characteristics,
            pe.OPTIONAL_HEADER.MajorLinkerVersion,
            pe.OPTIONAL_HEADER.MinorLinkerVersion,
            pe.OPTIONAL_HEADER.SizeOfCode,
            pe.OPTIONAL_HEADER.SizeOfInitializedData,
            pe.OPTIONAL_HEADER.SizeOfUninitializedData,
            pe.OPTIONAL_HEADER.AddressOfEntryPoint,
            pe.OPTIONAL_HEADER.BaseOfCode,
            pe.OPTIONAL_HEADER.ImageBase,
            pe.OPTIONAL_HEADER.SectionAlignment,
            pe.OPTIONAL_HEADER.FileAlignment,
            pe.OPTIONAL_HEADER.MajorOperatingSystemVersion,
            pe.OPTIONAL_HEADER.MinorOperatingSystemVersion,
            pe.OPTIONAL_HEADER.MajorImageVersion,
            pe.OPTIONAL_HEADER.MinorImageVersion,
            pe.OPTIONAL_HEADER.MajorSubsystemVersion,
            pe.OPTIONAL_HEADER.MinorSubsystemVersion,
            pe.OPTIONAL_HEADER.SizeOfImage,
            pe.OPTIONAL_HEADER.SizeOfHeaders,
            pe.OPTIONAL_HEADER.CheckSum,
            pe.OPTIONAL_HEADER.Subsystem,
            pe.OPTIONAL_HEADER.DllCharacteristics,
            pe.OPTIONAL_HEADER.SizeOfStackReserve,
            pe.OPTIONAL_HEADER.SizeOfStackCommit,
            pe.OPTIONAL_HEADER.SizeOfHeapReserve,
            pe.OPTIONAL_HEADER.SizeOfHeapCommit,
            pe.OPTIONAL_HEADER.LoaderFlags,
            pe.OPTIONAL_HEADER.NumberOfRvaAndSizes,
            len(pe.sections),  # Number of sections
        ]
        # Section features
        sections_entropy = [section.get_entropy() for section in pe.sections]
        features.extend([
            np.mean(sections_entropy),
            np.min(sections_entropy),
            np.max(sections_entropy),
            np.mean([section.SizeOfRawData for section in pe.sections]),
            np.min([section.SizeOfRawData for section in pe.sections]),
            np.max([section.SizeOfRawData for section in pe.sections]),
            np.mean([section.Misc_VirtualSize for section in pe.sections]),
            np.min([section.Misc_VirtualSize for section in pe.sections]),
            np.max([section.Misc_VirtualSize for section in pe.sections]),
        ])
        # Import features
        imports_nb_dll = len(pe.DIRECTORY_ENTRY_IMPORT) if hasattr(pe, 'DIRECTORY_ENTRY_IMPORT') else 0
        imports_nb = sum([len(entry.imports) for entry in pe.DIRECTORY_ENTRY_IMPORT]) if hasattr(pe, 'DIRECTORY_ENTRY_IMPORT') else 0
        imports_nb_ordinal = sum([len([imp for imp in entry.imports if imp.ordinal]) for entry in pe.DIRECTORY_ENTRY_IMPORT]) if hasattr(pe, 'DIRECTORY_ENTRY_IMPORT') else 0
```

```
        features.extend([imports_nb_dll, imports_nb, imports_nb_ordinal])
        # Export features
        export_nb   =   len(pe.DIRECTORY_ENTRY_EXPORT.symbols)   if
hasattr(pe, 'DIRECTORY_ENTRY_EXPORT') else 0
        features.append(export_nb)
        # Resource features
        resources = []
        if hasattr(pe, 'DIRECTORY_ENTRY_RESOURCE'):
            for entry in pe.DIRECTORY_ENTRY_RESOURCE.entries:
                if hasattr(entry, 'directory'):
                    for res in entry.directory.entries:
                        data_rva = res.data.struct.OffsetToData
                        size = res.data.struct.Size
                        data = pe.get_data(data_rva, size)
                        entropy = entropy_calculate(data)
                        resources.append((entropy, size))
        if resources:
            resources_mean_entropy = np.mean([res[0] for res in resources])
            resources_min_entropy = np.min([res[0] for res in resources])
            resources_max_entropy = np.max([res[0] for res in resources])
            resources_mean_size = np.mean([res[1] for res in resources])
            resources_min_size = np.min([res[1] for res in resources])
            resources_max_size = np.max([res[1] for res in resources])
        else:
            resources_mean_entropy      =      resources_min_entropy      =
resources_max_entropy = 0
         resources_mean_size = resources_min_size = resources_max_size = 0
        features.extend([
            len(pe.DIRECTORY_ENTRY_RESOURCE.entries)   if   hasattr(pe,
'DIRECTORY_ENTRY_RESOURCE') else 0,
            resources_mean_entropy,
            resources_min_entropy,
            resources_max_entropy,
            resources_mean_size,
            resources_min_size,
            resources_max_size,
        ])
        # Load configuration size
        load_configuration_size = pe.DIRECTORY_ENTRY_LOAD_CONFIG.struct.Size if hasattr(pe, 'DIRECTORY_ENTRY_LOAD_CONFIG') else 0
        features.append(load_configuration_size)
        # Version information size
        version_information_size = len(pe.VS_FIXEDFILEINFO) if hasattr(pe, 'VS_FIXEDFILEINFO') else 0
        features.append(version_information_size)
        return features
    except Exception as e:
        print(f"Error extracting features: {e}")
        return [0] * 53  # Ensure the feature vector matches the training data
def entropy_calculate(data):
    """Calculate the entropy of a given data."""
    if not data:
        return 0
    entropy = 0
    data_len = len(data)
    freq = {}
    for byte in data:
        if byte in freq:
            freq[byte] += 1
        else:
            freq[byte] = 1
    for byte in freq:
        p_x = freq[byte] / data_len
        entropy -= p_x * np.log2(p_x)
    return entropy
```

fileCompress.py

```
import lzma
import shutil
import os
from core.settings import MEDIA_ROOT
def CompressFile(filename):
    if not filename:
        return False
    filepath = os.path.join(MEDIA_ROOT, 'UploadFiles', filename)
    if not os.path.exists(filepath):
        return False
    # Define the path for the compressed file
    compressed_dir = os.path.join(MEDIA_ROOT, 'CompressedFiles')

    # Ensure the directory exists
    if not os.path.exists(compressed_dir):
        os.makedirs(compressed_dir)
    compressed_filepath = os.path.join(compressed_dir, f"{filename}.xz")
    try:
        # Compress the file using lzma with maximum compression level
        with open(filepath, 'rb') as f_in:
            with lzma.open(compressed_filepath, 'wb', preset=9 |
lzma.PRESET_EXTREME) as f_out:
                shutil.copyfileobj(f_in, f_out)
        return f"{filename}.xz"
    except Exception as e:
        print(f"An error occurred while compressing the file: {e}")
        return False
```

References

[1] T. S. K. ,. A. Y. A.Abdo, "A hybrid approach to secure and compress data streams in cloud computing environment," Science Direct, March 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157824000880.

[2] "How it Works," Virus Total, [Online]. Available: https://docs.virustotal.com/docs/how-it-works.

[3] V. Pasca, "Hybrid Analysis Blog," Hybrid Analysis, 2024. [Online]. Available: https://hybrid-analysis.blogspot.com/.

[4] "Malware Bytes," Malware Bytes, [Online]. Available: https://www.threatdown.com/about-us/.

[5] "7 Zip," 7 Zip, [Online]. Available: https://www.7-zip.org/faq.html.

[6] "What Is a Feasibility Study? How to Conduct One for Your Project," Project Manager, April 2023. [Online]. Available: https://www.projectmanager.com/training/how-to-conduct-a-feasibility-study.

[7] "Decision Tree Algorithm," Geeks For Geeks, May 2024. [Online]. Available: https://www.geeksforgeeks.org/decision-tree-algorithms/.

[8] "Lempel-Ziv-Markov chain algorithm," Linux Reviews, June 2019. [Online]. Available: https://linuxreviews.org/Lempel-Ziv-Markov_chain_algorithm.

[9] "Django Introduction," W3 School, [Online]. Available: https://www.w3schools.com/django/django_intro.php.

[10] "Chat GPT," Open AI, [Online]. Available: https://chatgpt.com/.