

# GOVERNMENT DEGREE COLLEGE, TUNI

## RANSOMWARE DETECTOR USING DEEP LEARNING

Report submitted to

DEPARTMENT OF COMPUTER SCIENCE

For the Award of the Degree of

**Master of Computer Science**

by

**NAKIREDDI VEERA MANIKANTA**

Reg.No. 2284229005



Under the esteemed guidance of

**Mr. Rasmi Ranjan Khansama**

**Lecturer in Computer Science**

DEPARTMENT OF COMPUTER SCIENCE

**GOVERNMENT DEGREE COLLEGE, TUNI**

Affiliated to Adikavi Nannaya University

KAKINADA DISTRICT

**2022 – 2024**

# **GOVERNMENT DEGREE COLLEGE, TUNI**

## **DECLARATION**

I declare that,

The work contained in this project is work done by me under the guidance of any supervisors. The work has not been submitted to any other institute for any degree or diploma. I have followed the guidelines provided by the university in preparing the report. I have conformed to the norms and guidelines given in the ethical code of conduct of the Institute. Whenever I have used materials from other sources, I have given due credit to them by citing them in the text of the report and giving their in the references.

**NAKIREDDI VEERA MANIKANTA**

(Signature of the Student)

**GOVERNMENT DEGREE COLLEGE, TUNI**  
**DEPARTMENT OF COMPUTER SCIENCE**



**CERTIFICATE**

This is to certify that this project report entitled, “**RANSOMWARE DETECTOR USING DEEP LEARNING**” is a bonafied work of Mr. **NAKIREDDI VEERA MANIKANTA**, Reg.No. **2284229005** submitted in a partial fulfilment of the requirements for the award of Degree of Master of Computer Science during the period **2022-2024**.

This work is carried out by him under my supervision and guidance and submitted to Department of Computer Science, Government Degree College, Tuni.

\_\_\_\_\_  
**INTERNAL GUIDE**

\_\_\_\_\_  
**HEAD OF THE DEPARTMENT**

\_\_\_\_\_  
**EXTERNAL EXAMINER**

## ACKNOWLEDGEMENT

I feel fortunate to pursue my **Master of Computer Science** from the campus **Government Degree College, Tuni**. It provided all the facilities in the areas of Computer Science.

It's a genuine pleasure to express my deep sense of thanks and gratitude to my mentor and project guide **Mr. Rasmi Ranjan Khansama, Lecturer in computer science** whose valuable guidance has been the ones that helped me patch this project and her instructions has served as the major contributor towards the completion of the project.

I profusely thank **Dr. Ch. LALITHA, Principal, Government of Degree College, Tuni**. for all the encouragement and support.

I also thank **Dr. Namballa Sridhar, Lecturer in Computer Science, Department of Computer Science** for the guidance throughout the academics.

I am extremely thankful to our **Mr. Rasmi Ranjan Khansama, Head of Department** who has been a source of inspiration for me throughout my project and for his valuable advices in making my project a success.

A great deal of thanks goes to **Review Committee Members** and the entire **Faculty of Government of Degree college, Tuni** for their excellent supervision and valuable suggestions.

I am always thankful to my family, friends and well-wishers for all their trust and constant support in the successful completion of my project.

## ABSTRACT

This project demonstrates a novel approach to detecting ransomware targeted at Microsoft Windows, taking files as input in '.exe' file extension, and returning a prediction of the file belonging to 1 of 3 classes: benign, generic malware, or ransomware. Ransomware is a type of malicious software that encrypts victim's data and demands payment in exchange for the decryption key. It has become a serious threat to organizations and individuals alike. The proposed system uses a Artificial Neural Network (ANN) algorithm to identify ransomware behaviour. The proposed system leverages the capabilities of ANN, a subset of machine learning algorithms that mimic the human brain's neural structure, to analyse patterns and characteristics indicative of ransomware behaviours data for training the ANN model comprises a diverse set of ransomware samples, benign applications, and system files to ensure generalization and accuracy. Features like file access patterns, network behaviour, and code execution are extracted, and a suitable feature selection method is employed to reduce dimensionality while retaining crucial information. The proposed ANN-based approach provides a flexible and adaptive solution capable of safeguarding critical data and systems from the ever-changing landscape of ransomware attacks.

**KEYWORDS:** Ransomware, Malware, Windows.

## INDEX

S.NO.	CONTENT	PAGE NO:
1.	<b>INTRODUCTION</b>	1-2
2.	<b>LITERATURE REVIEW</b>	3-4
3.	<b>PROBLEM SPECIFICATIONS</b>	
	3.1 Existing System	5
	3.2 Proposed System	6
4.	<b>SYSTEM SPECIFICATIONS</b>	
	4.1. Functional Requirements	7
	4.2. Non Functional Requirements	8
	4.3. Hardware Configuration	9
	4.4. Software Configuration	9
5.	<b>SYSTEM DESIGN</b>	
	5.1. Modules	10-11
	5.2. UML Diagrams	12
	5.2.1. Use case Diagram	13
	5.2.2. Class Diagram	14
	5.2.3. Sequence Diagram	15
	5.2.4. Activity Diagram	16
6.	<b>SYSTEM IMPLEMENTATION</b>	

	6.1. Software Installation	17
	6.1.1. Installing Python	17
	6.1.2. Installing PyCharm	18-19
	6.2. Introduction to Python	20-25
	6.3. System Study	25
	6.3.1. Feasibility Study	25
	6.3.2. Economical Feasibility	26
	6.3.3. Technical feasibility	26
	6.3.4. Social Feasibility	26
	6.4 Algorithm Description	27-28
7.	<b>CODING</b>	
	7.1. Importing Packages	29
	7.2. Modeling	30-35
	7.2.1. Defining Opcode Model	30-31
	7.2.2. Defining the strings as grayscale images model	31-32
	7.2.3. Ensemble Code	32-33
	7.3. Preprocessing	33
	7.4. Prediction	34-35
8.	<b>RESULTS</b>	36-40
9.	<b>SYSTEM TESTING</b>	41-43
10.	<b>CONCLUSION</b>	44
11.	<b>FUTURE SCOPE</b>	45
12.	<b>REFERENCES</b>	46

## LIST OF FIGURES

<b>Fig No.</b>	<b>Contents</b>	<b>Page No</b>
5.2.1	Use Case Diagram	13
5.2.2	Class Diagram	14
5.2.3	Sequence Diagram	15
5.2.4	Activity Diagram	16
8.1	Home Page	36
8.2	Registration Page	37
8.3	Login Page	38
8.4	Choose Folder Page	39
8.5	Scanning Page	39
8.6	Result	40



# CHAPTER-1

## INTRODUCTION

Ransomware is a malware type that is designed to prevent or reduce access a user has to their device, operating system, or files. Ransomware is typically found in the forms of locker ransomware and crypto-ransomware. Locker ransomware displays a lock screen that prevents the victim from accessing their computers, often pretending to be law



enforcement demanding monetary payment in return for access to the computer.

Crypto-ransomware encrypts key files on a user's system, using complex encryption schemes and demand fees, usually in the form of crypto currency to decrypt the victim's files. The decision to evaluate machine learning and deep learning approaches as opposed to other non-machine learning-based approaches was taken because of their adaptability and strong ability to detect unseen samples of ransomware malware. Non-learning approaches tend to lack the ability to adapt or be retrained to a new concept quickly. These approaches would take significantly more time to recalibrate. We have an interest in the possible wide-scale integration of these solutions in IoT (Internet of Things) to prevent the infection of IoT devices. In today's digital landscape, the threat of ransomware looms large, targeting individuals, businesses, and government entities alike. Ransomware is a malicious software that encrypts critical data and demands a ransom for

its release, causing significant financial losses and operational disruptions. Traditional security measures struggle to keep pace with the constantly evolving ransomware variants, necessitating innovative and efficient solutions. In this context, Artificial Neural Networks (ANNs) have emerged as a promising tool in cybersecurity, offering the potential to detect and mitigate ransomware attacks proactively. This research aims to introduce a robust ransomware detection system based on Artificial Neural Networks, presenting a proactive approach to safeguarding against these cyber threats. Artificial Neural Networks, inspired by the human brain's neural architecture, are a subset of machine learning algorithms designed to recognize patterns and relationships within complex data. ANNs consist of interconnected nodes organized into layers, each layer contributing to the network's ability to learn and make predictions. By training on extensive datasets, ANNs can automatically learn intricate features and behaviours associated with ransomware, enabling them to detect malicious activities with high accuracy. This adaptability and capability to handle non-linear relationships make ANNs well-suited for addressing the dynamic nature of ransomware attacks. Building an effective ransomware detection system using ANNs involves overcoming specific challenges unique to the cybersecurity domain. Ransomware authors constantly modify their code to evade detection, leading to a continuous arms race between attackers and defenders. As a result, the training data for ANNs must be comprehensive and updated regularly to reflect the latest ransomware variants accurately. Moreover, due to the criticality of real-time detection, the ransomware detection model must operate efficiently and swiftly without compromising accuracy. Addressing these challenges is vital to developing an ANN-based ransomware detector that can effectively protect against emerging threats. The system will continuously monitor system activities and network traffic, promptly identifying and flagging ransomware activity as soon as it occurs. By employing deep neural architectures, the system will analyse diverse features and behaviours of potential ransomware, ensuring comprehensive coverage and improved detection rates. To keep pace with evolving ransomware threats, the system will be regularly updated with the latest datasets, enhancing its ability to recognize new ransomware variants. The system will be fine-tuned to minimize false positives, reducing unnecessary alerts and allowing security teams to focus on genuine threats. The research on building a ransomware detection system using Artificial Neural Networks holds

significant importance in the realm of cybersecurity. By leveraging the capabilities of ANNs, this system can identify ransomware attacks proactively, thwarting the encryption process before substantial damage occurs. The proposed solution not only helps organizations and individuals protect their critical data but also strengthens the overall resilience against ransomware attacks in the digital ecosystem. Furthermore, the knowledge gained from this research can be extended to other cyber security applications, broadening the spectrum of defence against a wide range of cyber threats.



## **CHAPTER-2**

### **LITERATURE REVIEW**

**[1] Baig, M.; Zavorsky, P.; Ruhl, R.; Lindskog, D. The study of evasion of packed PE from static detection. In Proceedings of the World Congress on Internet Security (World CIS), Guelph, Ontario, 10–12 June 2012; pp. 99–104.**

Static detection of packed portable executables (PEs) relies primarily on structural properties of the packed PE, and also on anomalies in the packed PE caused by packing tools. This paper outlines weaknesses in this method of detection will show that these structural properties and anomalies are contingent features of the executable, and can be more or less easily modified to evade static detection[1].

**[2] Zakaria, W.Z.A.; Mohd, M.F.A.O.; Ariffin, A.F.M. The Rise of Ransomware. In Proceedings of the 2017 International Conference on Software and e-Business, ICSEB 2017, Hong Kong, 28–30 December 2017; pp. 66–70**

Ransomware continues to be one of the most crucial cyber threats and is actively threatening IT users around the world. In recent years, it has become a phenomenon and traumatic threat to individuals, governments and organizations. Ransomwares not only penalized computational operations, it also mercilessly extorts huge amount of money from the victims if the victims want to regain back access to the system and files. As such, the cybercriminals are making millions of profits and keep on spreading new variants of ransomware. This paper discusses about ransomware and some related works in fighting this threat[2].

**[3] Narayanan Balkrishnan and Dr. Govindarajan Muthukumarasamy. Crop production Ensemble Machine Learning model for prediction. The 11th ACM International Conference on Management of Digital EcoSystems (2019).**

The manual system of predicting future forecasting with large number of cases is very time consuming and usually ends up with various classes clashing either... To

overcome all these problems, propose to make a web application system. The system will take various inputs like details of cases, recoveries and deaths, depending upon these inputs it will generate a possible response, making optimal predictions of all resources in a way that will best suit any of constraints. [3]

**[4] Kolter, J.Z.; Maloof, M.A. Learning to detect and classify malicious executables in the wild. J. Mach. Learn. Res. 2006, 7, 2721–2744.**

We described the use of machine learning and data mining to detect and classify malicious executables as they appear in the wild. We gathered 1,971 benign and 1,651 malicious executables and encoded each as a training example using n-grams of byte codes as features. Such processing resulted in more than 255 million distinct n-grams. After selecting the most relevant n-grams for prediction, we evaluated a variety of inductive methods, including naive Bayes, decision trees, support vector machines, and boosting. Ultimately, boosted decision trees outperformed other methods with an area under the ROC curve of 0.996. Results suggest that our methodology will scale to larger collections of executables. We also evaluated how well the methods classified executables based on the function of their payload, such as opening a backdoor and mass-mailing. Areas under the ROC curve for detecting payload function were in the neighbourhood of 0.9, which were smaller than those for the detection task. However, we attribute this drop in performance to fewer training examples and to the challenge of obtaining properly labelled examples, rather than to a failing of the methodology or to some inherent difficulty of the classification task. Finally, we applied detectors to 291 malicious executables discovered after we gathered our original collection, and boosted decision trees achieved a true-positive rate of 0.98 for a desired false-positive rate of 0.05. This result is particularly important, for it suggests that our methodology could be used as the basis for an operational system for detecting previously undiscovered malicious executables[4]

## **CHAPTER-3**

### **PROBLEM SPECIFICATION**

#### **3.1. EXISTING SYSTEM:**

Traditional Machine Learning algorithms basically use observations to estimate predictions. Traditional practices fail to detect this type of ransomwares. Machine Learning have different types of algorithms and each has its own structure of working procedure which contains different intuitions. In these intuitions, models are working on different procedures in different ways and also delivers high and less accuracies. The major challenge is to create a model for them so that no one have less accuracy. In this existing system we used ANN & Logistic regression which results less accuracy. Such a models won't be able to match with our requirements and it also consumes more time.

#### **DISADVANTAGES:**

- Low efficiency.
- Time consuming.
- High complexities.
- Resources consuming

#### **3.2.PROPOSED METHOD**

I have proposed this application that can be considered a useful system since it helps to reduce the limitations obtained from KNN and Logistic Regression. By providing support through the regression analysis, it can be able to generate best results for attributes without any overlap. The system is developed in a Flask based Python environment. MySQL is used for database management and the models involved in this application are artificial neural network ANN. In the proposed system, we aim to develop a ransomware

detector utilizing Artificial Neural Networks (ANN) for enhanced cybersecurity measures. Ransomware has become a significant threat to individuals, businesses, and organizations worldwide, causing data breaches, financial losses, and operational disruptions. To combat this menace effectively, we leverage the capabilities of ANN, which is a powerful machine learning technique inspired by the human brain's neural networks. The ransomware detector will be designed to analyse and classify incoming data streams, such as files, emails, and network traffic, to identify potential ransomware attacks. The proposed ransomware detector using ANN is expected to enhance the overall cyber security posture of the target system, providing proactive defence against ransomware attacks and bolstering protection for sensitive data and critical infrastructure. By leveraging the power of machine learning, the system can adapt to new ransomware variants and emerging attack vectors, making it an essential tool in the fight against cyber threats.

#### **ADVANTAGES:**

- High efficiency.
- Time Saving.
- Inexpensive.
- Low complexities.

## **CHAPTER-4**

### **SYSTEM SPECIFICATIONS**

#### **4.1. FUNCTIONAL REQUIREMENTS:**

- **Data Collection:** The system collect representative datasets that include both benign and ransomware samples.
- **Data Preprocessing:** The system preprocess the collected data effectively. This includes tasks like normalizing the data, cleaning noise or irrelevant information, handling missing values.
- **Feature Extraction:** The system is capable of extracting meaningful features from the pre-processed data. The features should capture important characteristics and behaviors that differentiate ransomware from benign activity
- **Training and Learning:** The system will provide functionalities for training and learning the neural network model using the pre-processed data. This includes tasks like splitting the dataset into training and validation sets.
- **Real-time Monitoring and Detection:** The system is capable of deploying the trained model for real-time monitoring and detection of ransomware activity. The system should continuously analyse incoming data and make timely predictions to identify potential ransomware attacks.

#### **4.2. NON FUNCTIONAL REQUIREMENTS**

- **Performance:** The system is efficient and capable of handling large volumes of data in real-time. It should be designed to process and analyse data and ensuring timely detection and response to ransomware threats.



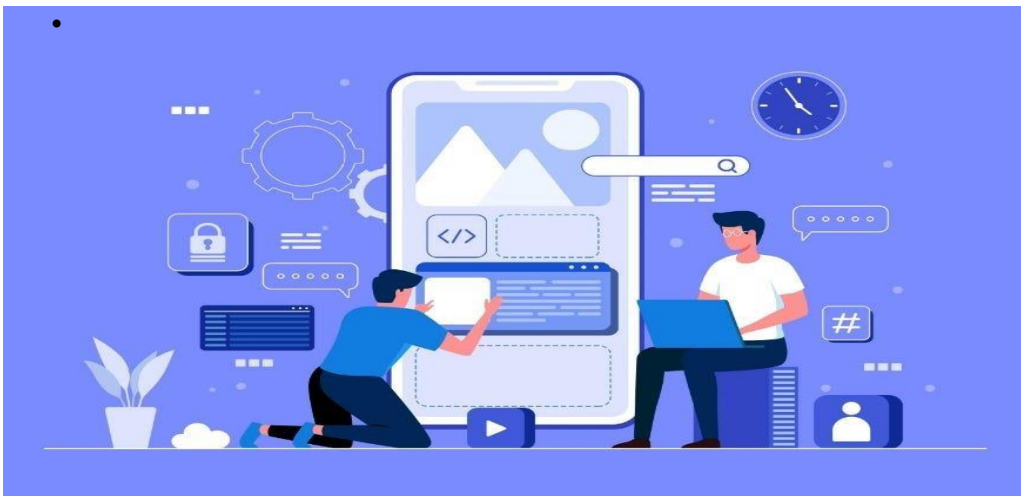
- **Accuracy:** The system will exhibit a high level of accuracy in detecting ransomware. The trained ANN models should demonstrate reliable performance in distinguishing between ransomware and benign activity.
- **Scalability:** The system is designed to scale effectively as the amount of data and the number of users increase. It is capable of handling a growing volume of data without compromising the detection accuracy.
- **Usability:** The system is user-friendly and intuitive, allowing security analysts or administrators to interact with and manage the ransomware detection system efficiently.
- **Performance Monitoring :** The system provide mechanisms for monitoring its own performance, generating reports on detection.

#### **4.3. HARDWARE CONFIGURATION:**

- Processor : I5/Intel Processor
- Ram : 8 GB
- Hard Disk : 256 GB

#### 4.4. SOFTWARE CONFIGURATION:

- Operating system : Windows 10
- IDE : PyCharm
- Technology : Python 3.6+
- Framework : Flask
- Database : MySQL



# **CHAPTER-5**

## **SYSTEM DESIGN**

### **5.1. MODULES**

#### **Data Collection:**

Responsible for gathering a diverse and representative dataset containing both normal and ransomware samples. This module may interact with various sources, such as file systems, emails, or network traffic logs.

#### **Data Preprocessing:**

Handles data cleaning, transformation, and normalization to prepare the dataset for training. It may involve removing noise, handling missing values, and converting data into a suitable format for the ANN model.

#### **Feature Extraction:**

Extracts relevant features from the pre-processed data to be used as input for the ANN. Feature extraction techniques may involve statistical, behavioural, or structural attributes of files, emails, or network packets.

#### **Artificial Neural Network:**

This is the core module that defines the architecture and implementation of the ANN model. It includes layers, activation functions, optimization algorithms, and any additional customization specific to ransomware detection.

#### **Training :**

Utilizes the prepared dataset to train the ANN model. This module applies supervised learning techniques, where the model learns to classify data into normal or ransomware categories based on the provided training labels.

**Evaluation:**

Evaluates the performance of the trained ANN model on separate validation and test datasets. It calculates various metrics such as accuracy, precision, recall, F1 score, and confusion matrices to assess the model's effectiveness.

**Real-Time Detection :**

Integrates the trained ANN model into the live system to perform real-time ransomware detection. It continuously monitors incoming data streams and makes predictions on potential ransomware activities.

**Alerting and Response :**

If the ransomware detector identifies a potential ransomware attack, this module triggers appropriate responses, such as generating alerts, blocking malicious activities, or notifying administrators.

**Logging and Reporting :**

Maintains logs of detected activities, model performance metrics, and any generated alerts for auditing and reporting purposes. It helps in monitoring and understanding the system's behaviour.

**User Interface:**

Provides a user-friendly interface for system configuration, monitoring, and interaction. It allows administrators to manage the ransomware detection system efficiently.

**Integration :**

Facilitates seamless integration with existing cybersecurity infrastructure or applications to enhance overall security capabilities.

## 5.2. UML DIAGRAMS

- UML stands for Unified Modelling Language. UML is a standardized general-purpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.
- The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.
- The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modelling and other non-software systems.
- The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems.
- The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

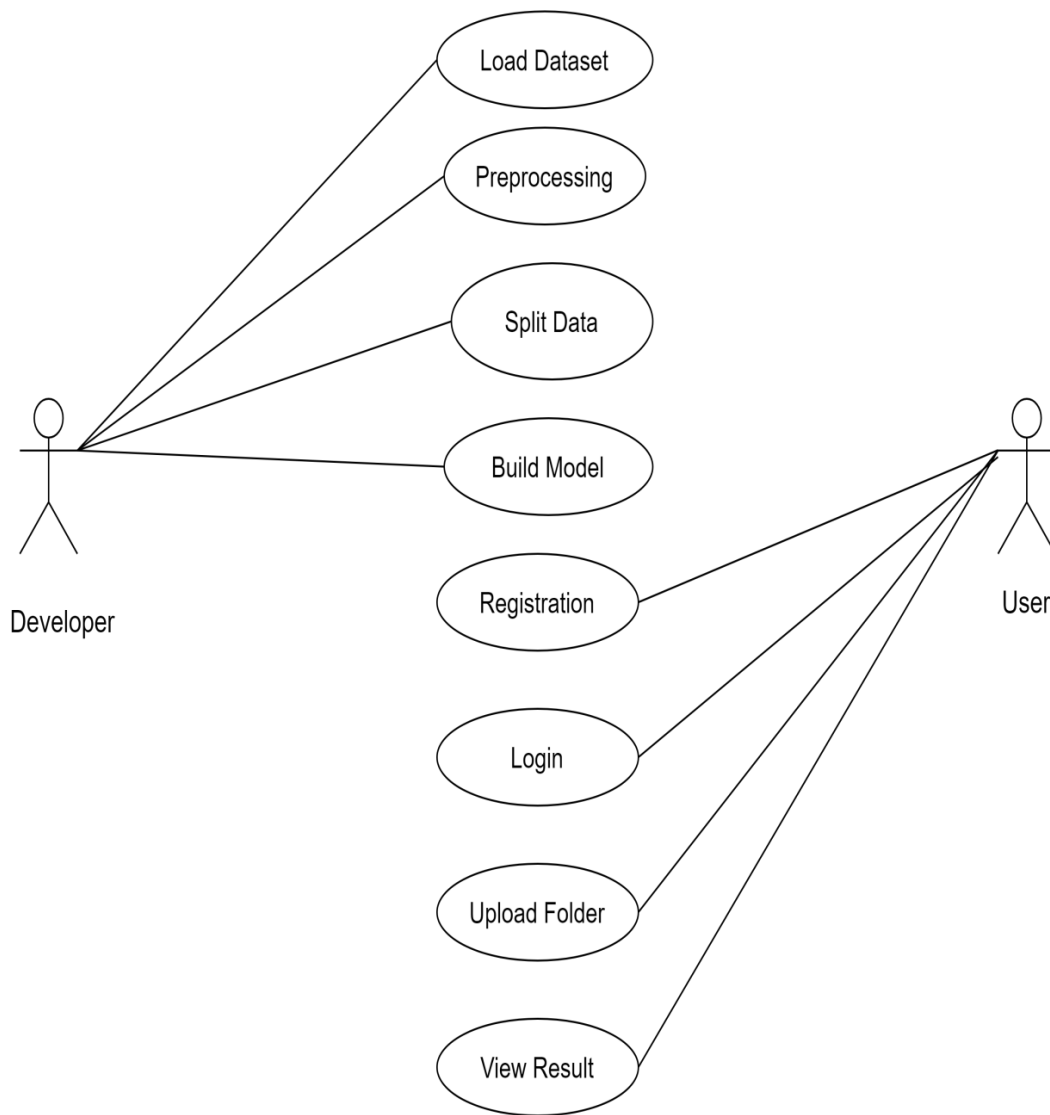
### GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modelling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modelling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

### 5.2.1. USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



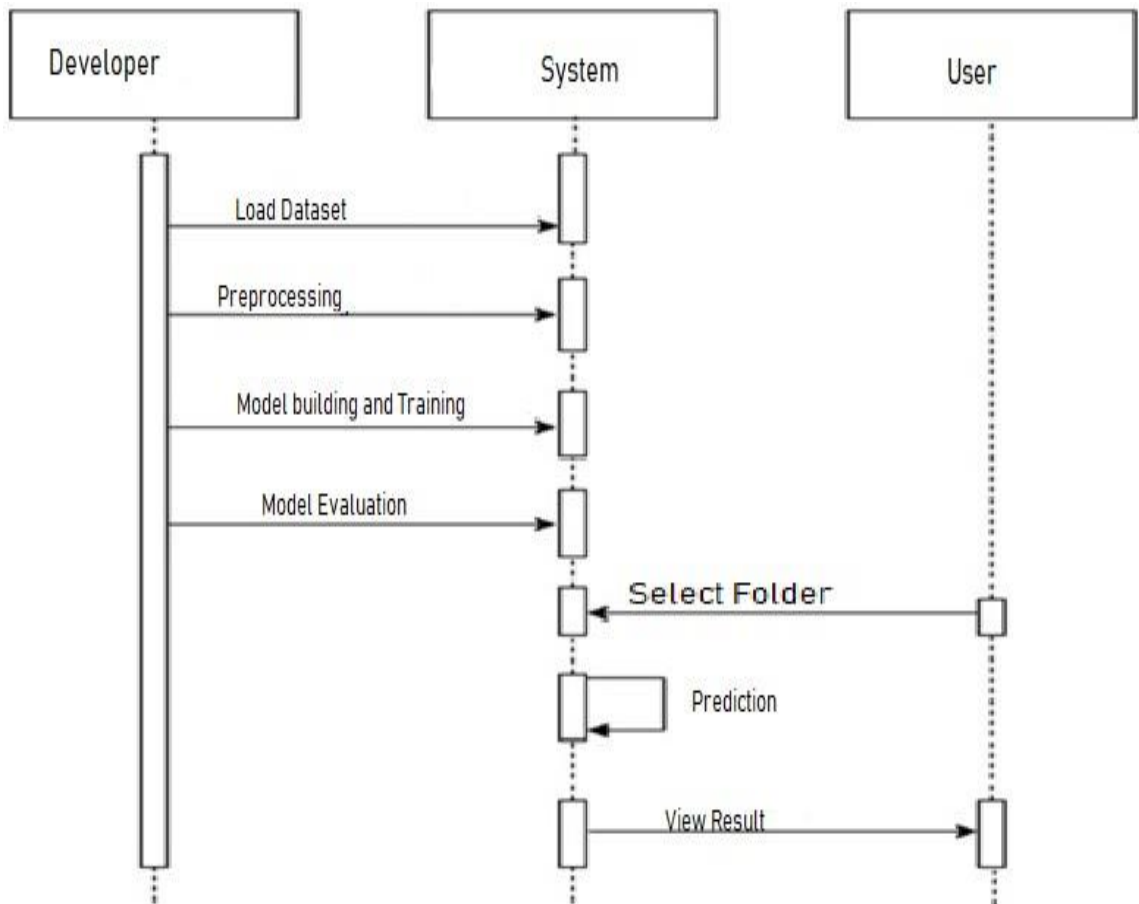
### 5.2.2. CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes.



### 5.2.3. SEQUENCE DIAGRAM:

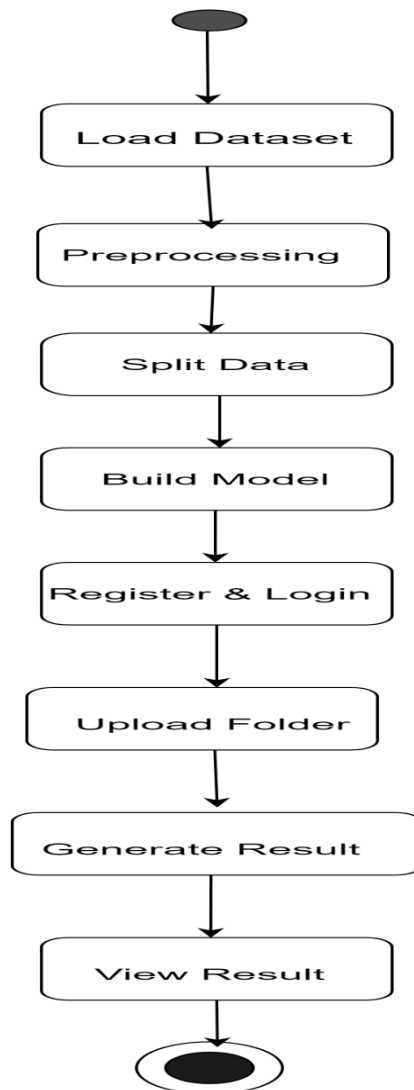
A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.





#### 5.2.4. ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



# CHAPTER-6

## SYSTEM IMPLEMENTATION

### 6.1. SOFTWARE INSTALLATION

#### 6.1.1. Installing Python:

1. To download and install Python visit the official website of Python <https://www.python.org/downloads/> and choose your version.



Once the download is complete, run the exe for install Python. Now click on Install Now.

2. You can see Python installing at this point.
3. When it finishes, you can see a screen that says the Setup was successful. Now click on "Close".

#### 6.1.2. Installing PyCharm:

1. To download PyCharm visit the website <https://www.jetbrains.com/help/pycharm/installation-guide.html> and Click the "DOWNLOAD" link under the Community Section.

# Download PyCharm

[Windows](#)

[Mac](#)

[Linux](#)

## Professional

For both Scientific and Web Python development. With HTML, JS, and SQL support.

Download

Free trial

## Community

For pure Python development

Download

Free, open-source

2. Once the download is complete, run the exe for install PyCharm. The setup wizard should have started. Click “Next”.
3. On the next screen, Change the installation path if required. Click “Next”.
4. On the next screen, you can create a desktop shortcut if you want and click on “Next”.
5. Choose the start menu folder. Keep selected JetBrains and click on “Install”.
6. Wait for the installation to finish.
7. Once installation finished, you should receive a message screen that PyCharm is installed. If you want to go ahead and run it, click the “Run PyCharm Community Edition” box first and click “Finish”.
8. After you click on "Finish," the Following screen will appear.



9. You need to install some packages to execute your project in a proper way.
10. Open the command prompt/ anaconda prompt or terminal as administrator.
11. The prompt will get open, with specified path, type “pip install package name” which you want to install (like NumPy, pandas, seaborn, scikit-learn, matplotlib.pyplot)

Ex: **pip install numpy**

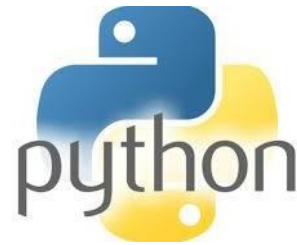
```
C:\Users\nvm14>pip install numpy
Collecting numpy
  Using cached numpy-1.26.4-cp312-cp312-win_amd64.whl.metadata (61 kB)
Using cached numpy-1.26.4-cp312-cp312-win_amd64.whl (15.5 MB)
Installing collected packages: numpy
Successfully installed numpy-1.26.4

C:\Users\nvm14>
```

## 6.2. INTRODUCTION TO PYTHON

Python is a high-level, versatile, and easy-to-learn programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python's design philosophy emphasizes code readability and a clear syntax, which makes it an excellent language for beginners and experienced programmers alike.

### Key Features of Python:



1. **Easy to Learn:** Python's syntax is straightforward and concise, making it accessible for beginners and reducing the learning curve.
2. **Versatile:** Python is a multipurpose language that can be used for a wide range of applications, including web development, data analysis, artificial intelligence, machine learning, scientific computing, automation, and more.
3. **Open-Source:** Python is open-source, meaning its source code is freely available, and developers worldwide can contribute to its improvement and expansion.
4. **Cross-Platform:** Python is a cross-platform language, which means you can write code on one operating system (e.g., Windows) and run it on another (e.g., macOS or Linux) without modifications.
5. **Large Standard Library:** Python comes with a comprehensive standard library that provides built-in modules and functions to perform various tasks, such as handling files, working with data structures, and implementing network protocols.
6. **Community and Ecosystem:** Python has a large and active community of developers who create and maintain a vast ecosystem of libraries and frameworks, making it easy to find tools for almost any programming need.
7. **Object-Oriented Programming (OOP):** Python supports object-oriented programming, allowing developers to create reusable and organized code using classes and objects.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below –

- It supports functional and structured programming methods as well as OOP.
  - It can be used as a scripting language or can be compiled to byte-code for building large applications.
  - It provides very high-level dynamic data types and supports dynamic type checking.
  - IT supports automatic garbage collection.
  - It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.
- 
- **Python is Interpreted** – Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
  - **Python is Interactive** – You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
  - **Python is Object-Oriented** – Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
  - **Python is a Beginner's Language** – Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

### **Some Python Libraries:**

- Pandas
- NumPy
- PyMySQL
- Scikit-learn

## Pandas:

- Pandas provide us with many Series and Data Frames. It allows you to easily organize, explore, represent, and manipulate data.
- Smart alignment and indexing featured in Pandas offer you a perfect organization and data labelling.
- Pandas has some special features that allow you to handle missing data or value with a proper measure.
- This package offers you such a clean code that even people with no or basic knowledge of programming can easily work with it.
- It provides a collection of built-in tools that allows you to both read and write data in different web services, data-structure, and databases as well.
- Pandas can support JSON, Excel, CSV, HDF5, and many other formats. In fact, you can merge different databases at a time with Pandas.



## NumPy:



- Arrays of NumPy offer modern mathematical implementations on huge amount of data. NumPy makes the execution of these projects much easier and hassle-free.

- NumPy provides masked arrays along with general array objects. It also comes with functionalities such as manipulation of logical shapes, discrete Fourier transform, general linear algebra, and many more.
- While you change the shape of any N-dimensional arrays, NumPy will create new arrays for that and delete the old ones.
- This python package provides useful tools for integration. You can easily integrate NumPy with programming languages such as C, C++, and Fortran code.
- NumPy provides such functionalities that are comparable to MATLAB. They both allow users to get faster with operations.

## PyMySQL:



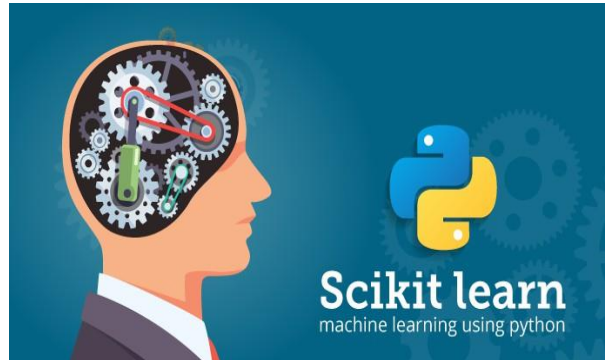
- PyMySQL is a database connector for Python, libraries to enable Python programs to talk to a MySQL server.
- Access to the port settings through Python properties.
- PyMySQL is a pure Python MySQL driver, first written as a rough port of the MySQL-Python driver.
- PyMySQL meets all of criterion for a driver.
- It is fully open source, hosted on Git hub, released on Pypi, is actively maintained.



- It is written in pure Python so is eventlet-monkeypatch compatible, and is fully Python 3 compatible.

## Scikit-Learn:

- The random module is a simple and efficient tool for predictive data analysis
- Accessible to everybody, and reusable in various contexts



- The library is focused on modeling data. It is not focused on loading, manipulating and summarizing data. For these features, refer to NumPy and Pandas.
- The functionality that scikit-learn provides include:
  - Regression, including Linear and Logistic Regression
  - Classification, including K-Nearest Neighbours
  - Clustering, including K-Means and K-Means++
  - Model selection
  - Preprocessing, including Min-Max Normalization

## Python modules:

Python allows us to store our code in files (also called modules). This is very useful for more serious programming, where we do not want to retype a long function definition from the very beginning just to change one mistake. In doing this, we are essentially defining our own modules, just like the modules defined already in the Python library.

To support this, Python has a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module; definitions from a module can be imported into other modules or into the main module.

## **Testing code:**

As indicated above, code is usually developed in a file using an editor.

To test the code, import it into a Python session and try to run it.

Usually there is an error, so you go back to the file, make a correction, and test again.

This process is repeated until you are satisfied that the code works. T

he entire process is known as the development cycle.

There are two types of errors that you will encounter. Syntax errors occur when the form of some command is invalid.

This happens when you make typing errors such as misspellings, or call something by the wrong name, and for many other reasons. Python will always give an error message for a syntax error.

## **Functions in Python:**

It is possible, and very useful, to define our own functions in Python. Generally speaking, if you need to do a calculation only once, then use the interpreter. But when you or others have need to perform a certain type of calculation many times, then define a function.

You use functions in programming to bundle a set of instructions that you want to use repeatedly or that, because of their complexity, are better self-contained in a sub-program and called when needed. That means that a function is a piece of code written to carry out a specified task.

To carry out that specific task, the function might or might not need multiple inputs. When the task is carried out, the function can or cannot return one or more values.

There are three types of functions in python:

help(),min(),print().

### **6.3. SYSTEM STUDY**

#### **6.3.1 FEASIBILITY STUDY**

The feasibility of the project is analysed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

#### **6.3.2. ECONOMICAL FEASIBILITY**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

### **6.3.3. TECHNICAL FEASIBILITY**

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

### **6.3.4. SOCIAL FEASIBILITY**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

## **6.4. Algorithm Description**

Artificial neural networks (ANNs), usually simply called neural networks (NNs), are computing systems vaguely inspired by the biological neural networks that constitute animal brains. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their

inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times. To detect ransomware using an Artificial Neural Network (ANN), one common approach involves using opcode sequences and grayscale images as input data.

### **Opcode Model:**

Opcode sequences are representations of the low-level instructions executed by a program or a file. In the context of ransomware detection, opcode sequences can be extracted from executable files or processes. Each executable file is disassembled, and its opcode sequences are obtained, forming a sequence of numerical values representing the executed instructions.

#### **How the Opcode Model works:**

- The opcode model processes the opcode sequences as input data for the ANN.
- The ANN is designed to learn patterns from legitimate executable files and ransomware samples.
- It learns to recognize specific opcode sequences or combinations of opcodes that are indicative of ransomware behaviour.
- The model can be trained using supervised learning, where opcode sequences are labelled as either ransomware or benign files.

### **Grayscale Model:**

Grayscale images can be generated by visualizing the binary content of files or network traffic. In the context of ransomware detection, grayscale images can represent visual patterns that differentiate ransomware files from benign files.

#### **How the Grayscale Model works:**

- Files, such as executables, can be converted into grayscale images by mapping binary data to pixel intensities (e.g., 0 for black and 255 for white).

- Specific visual patterns may emerge when analyzing ransomware files as grayscale images.
- The grayscale model takes these images as input data for the ANN.
- Similar to the opcode model, the ANN learns to differentiate between ransomware and benign files based on the visual patterns present in the grayscale images.

### **Combining the Models:**

- The opcode and grayscale models can be combined into a unified ANN architecture.
- Input data could include both the opcode sequences and grayscale representations of the executable files.
- By training on both types of data, the model can learn more comprehensive representations of ransomware behaviour and increase detection accuracy.
- The two modalities might also provide complementary information, improving the overall performance of the detector.

## CHAPTER-7

### CODE IMPLEMENTATION

#### 7.1. Importing Packages

```
import os, math, string, pefile, time, threading
import tkinter as tk
import NumPy as np
from capstone import *
from tensorflow.keras.models import Sequential, Model
from keras.models import Input
from tensorflow.keras import layers, preprocessing
from tensorflow.keras.utils import Sequence
from sklearn.utils import shuffle
from tkinter import messagebox
from tkinter.filedialog import askopenfilenames, askdirectory
from tkinter.ttk import Progressbar
import getpass
```

## 7.2. Modeling

### 7.2.1. Defining the Opcode Model:

```
opModel = Sequential(name="opcodeModel")

opModel.add(layers.InputLayer(input_shape=(50,)))
opModel.add(layers.Dense(256, activation='relu'))
opModel.add(layers.BatchNormalization())
opModel.add(layers.Dense(128, activation='relu'))
opModel.add(layers.BatchNormalization())
opModel.add(layers.Dense(64, activation='relu'))
opModel.add(layers.BatchNormalization())
opModel.add(layers.Dense(32, activation='relu'))
opModel.add(layers.BatchNormalization())
opModel.add(layers.Dense(16, activation='relu'))
opModel.add(layers.BatchNormalization())
opModel.add(layers.Dense(3, activation='softmax'))

opModel.load_weights("weights-improvement-574-0.85.hdf5")

opModel.compile(optimizer="rmsprop",
                loss='categorical_crossentropy',
                metrics=['accuracy'])

class histSequence(Sequence):

    def __init__(self, x, y, batch_size):
        self.x, self.y = shuffle(x, y)
        self.batch_size = batch_size

    def __len__(self):
        return math.ceil(len(self.x) / self.batch_size)

    def __getitem__(self, idx):
        batch_x = self.x[idx * self.batch_size:(idx + 1) *
```



```

        self.batch_size]
        batch_y = self.y[idx * self.batch_size:(idx + 1) *
        self.batch_size]

    return np.array([
        np.load(file_name)
        for file_name in batch_x]), np.array(batch_y)

    def on_epoch_end(self):
        pass

class histSequenceVal(histSequence):

    def __init__(self, x, y, batch_size):
        self.x, self.y = x, y
        self.batch_size = batch_size

```

### 7.2.2. Defining the strings as grayscale images model

```

model = Sequential(name="stringsAsGreyscaleModel")

model.add(layers.InputLayer(input_shape=(100, 100, 1)))
model.add(layers.SpatialDropout2D(rate=0.2))
model.add(layers.Conv2D(32, kernel_size=3, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.SpatialDropout2D(rate=0.1))
model.add(layers.Conv2D(16, kernel_size=3, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.SpatialDropout2D(rate=0.1))
model.add(layers.Flatten())
model.add(layers.Dense(3, activation='softmax'))

class hashCorpusSequence(Sequence):

    def __init__(self, x, y, batch_size):
        self.x, self.y = shuffle(x, y)
        self.batch_size = batch_size

```

```

def __len__(self):
    return math.ceil(len(self.x) / self.batch_size)

def __getitem__(self, idx):
    batch_x = self.x[idx * self.batch_size:(idx + 1) *
self.batch_size]
    batch_y = self.y[idx * self.batch_size:(idx + 1) *
self.batch_size]

    return np.array([
        np rint(((np.load(file_name) - np.min(np.load(file_name))) /
        (np.max(np.load(file_name)) - np.min(np.load(file_name)))) * 255).astype(int)
        for file_name in batch_x]), np.array(batch_y)

def on_epoch_end(self):
    pass

class hashCorpusSequenceVal(hashCorpusSequence):

def __init__(self, x, y, batch_size):
    self.x, self.y = x, y
    self.batch_size = batch_size

model.load_weights("weights-improvement-04-0.72.hdf5")

model.compile(optimizer="adamax",
              loss='categorical_crossentropy',
              metrics=['accuracy'])

```

### 7.2.3. Ensemble code:

```

def ensemble(models, model_inputs):
    outputs = [models[0](model_inputs[0]), models[1](model_inputs[1])]
    y = layers.average(outputs)
    modelEns = Model(model_inputs, y, name='ensemble')

```

```
return modelEns
```

```
models = [opModel, model]
model_inputs = [Input(shape=(50,)), Input(shape=(100, 100, 1))]
modelEns = ensemble(models, model_inputs)
modelEns.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])
```

### 7.3. Preprocessing:

```
def strings(filename, min=4):
    with open(filename, errors="ignore", encoding="utf-8") as f:
        result = ""
        for c in f.read():
            if c in string.printable:
                result += c
            continue
        if len(result) >= min:
            yield result
        result = ""
        if len(result) >= min:
            yield result
```

### 7.4. Prediction:

```
def predictPEs(pePaths):
    classNames = ["benign", "malware", "ransomware"]
    pePredictions = { }
```

```

    count = 0
    for pePath in pePaths:
        x1 = preprocessPEs(pePaths)[count][0].reshape(1, 50)
        x2 = preprocessPEs(pePaths)[count][1].reshape(1, 100, 100, 1)
        count += 1
        pePredictions[pePath] = classNames[np.argmax(modelEns.predict(x=[x1, x2]))]

    return pePredictions

def executeAntivirus(root):
    # tkRoot = tk.Tk()
    tkRoot = root
    tkRoot.title("Scanning files...")
    tkRoot.withdraw()
    tkRoot.protocol("WM_DELETE_WINDOW", quit)
    w = tkRoot.winfo_screenwidth()
    h = tkRoot.winfo_screenheight()
    size = tuple(int(pos) for pos in tkRoot.geometry().split('+')[0].split('x'))
    x = w / 2 - size[0] / 2
    y = h / 2 - size[1] / 2
    tkRoot.geometry("300x1+{ }+{ }".format(round(x) - 150, round(y)))

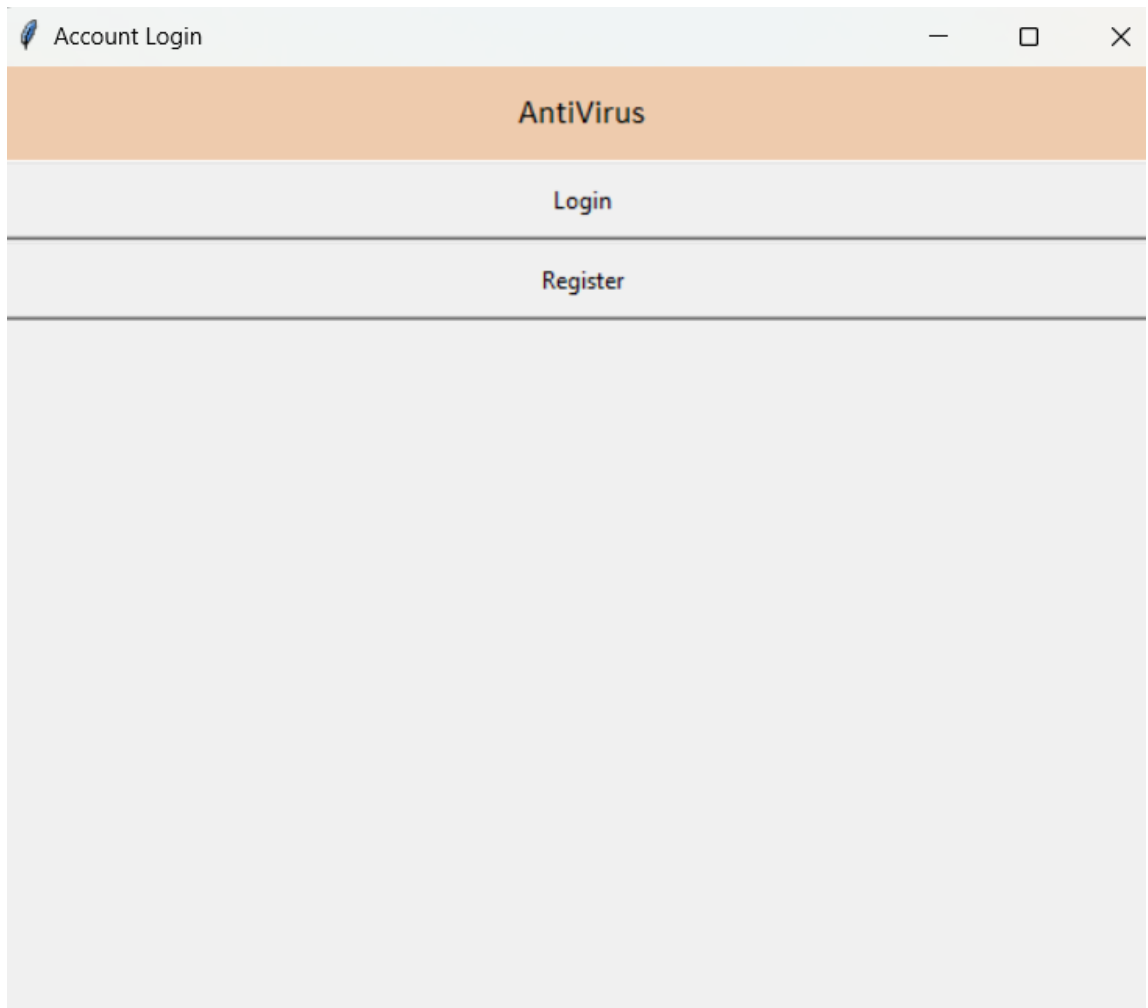
    while True:
        try:
            user = getpass.getuser()
            fPath = askdirectory(initialdir='c:/Users/%s' % user)
            print(fPath)
            print(os.listdir(fPath))
            fPath_ls = os.listdir(fPath)
            pePaths = [os.path.join(fPath, path) for path in fPath_ls if path[-4:] == ".exe"]
            print("PePaths", pePaths)
            tkRoot.update()
            tkRoot.deiconify()
            preds = predictPEs(pePaths)
            if len(preds) > 0:
                classificationsStr = ""
                for key in preds.keys():
                    classificationsStr += "" + key + "" + " detected as " + preds[key] + "\n\n"
            tkRoot.withdraw()

```

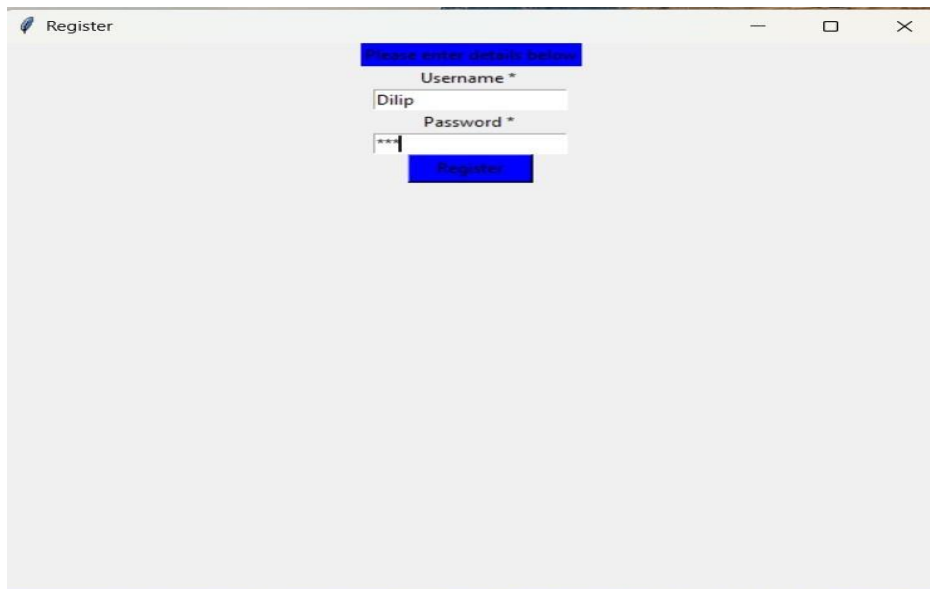
```
        messagebox.showinfo("Detections", classificationsStr)
    else:
        quit()
except Exception as e:
    messagebox.showerror("Error", "Error: " + str(e) + "\nPlease try again...")
```

## 8. RESULTS

### 8.1. Home:

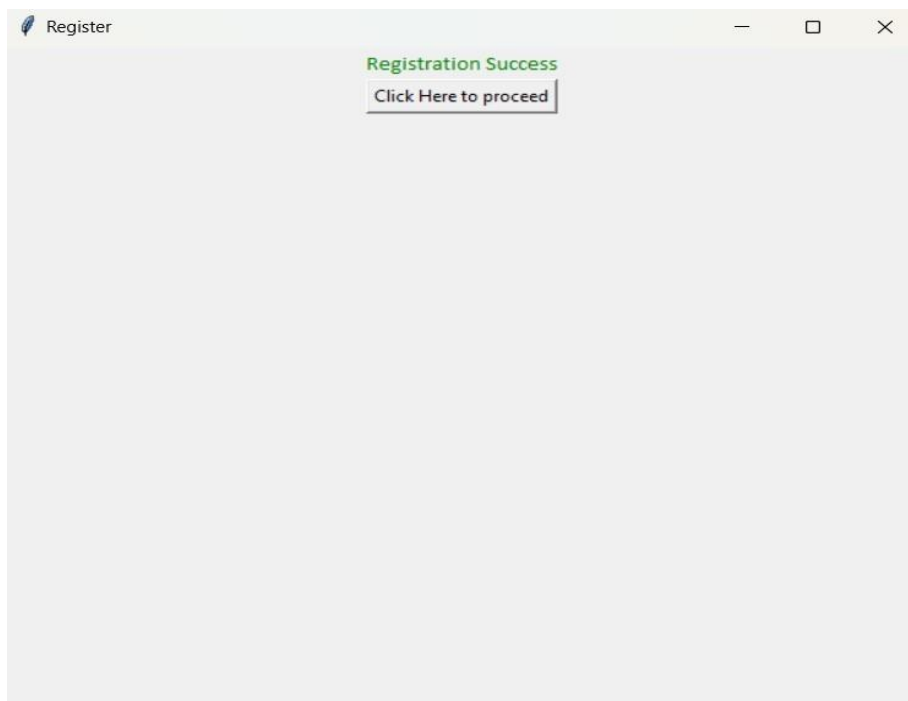


## 8.2. Registration:



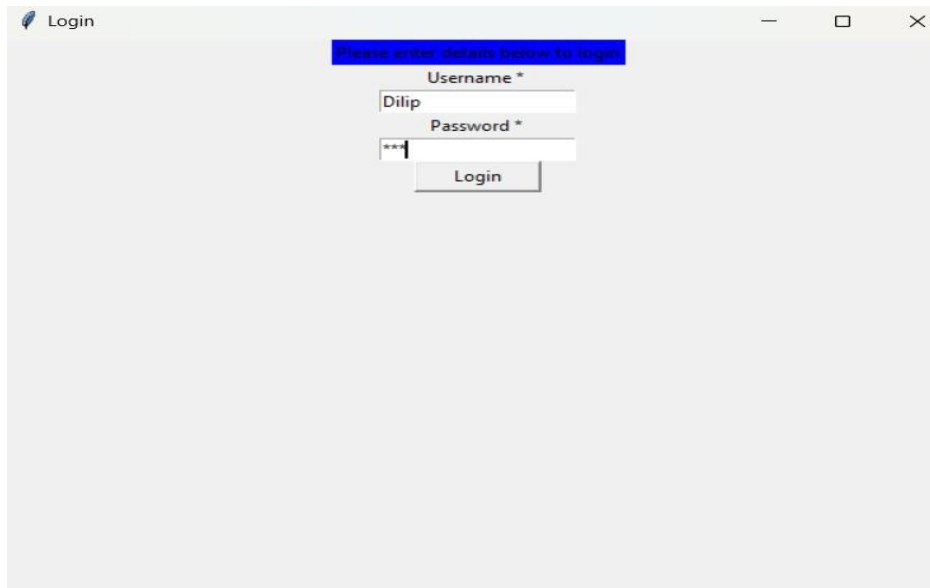
The screenshot shows a web browser window with the title 'Register'. The page has a light gray background. At the top, there is a blue banner with the text 'Please enter details below'. Below the banner, there are two input fields. The first is labeled 'Username \*' and contains the text 'Dilip'. The second is labeled 'Password \*' and contains three asterisks '\*\*\*'. Below the password field, there is a blue button with the text 'Register'.

## 8.3. Registration Success:



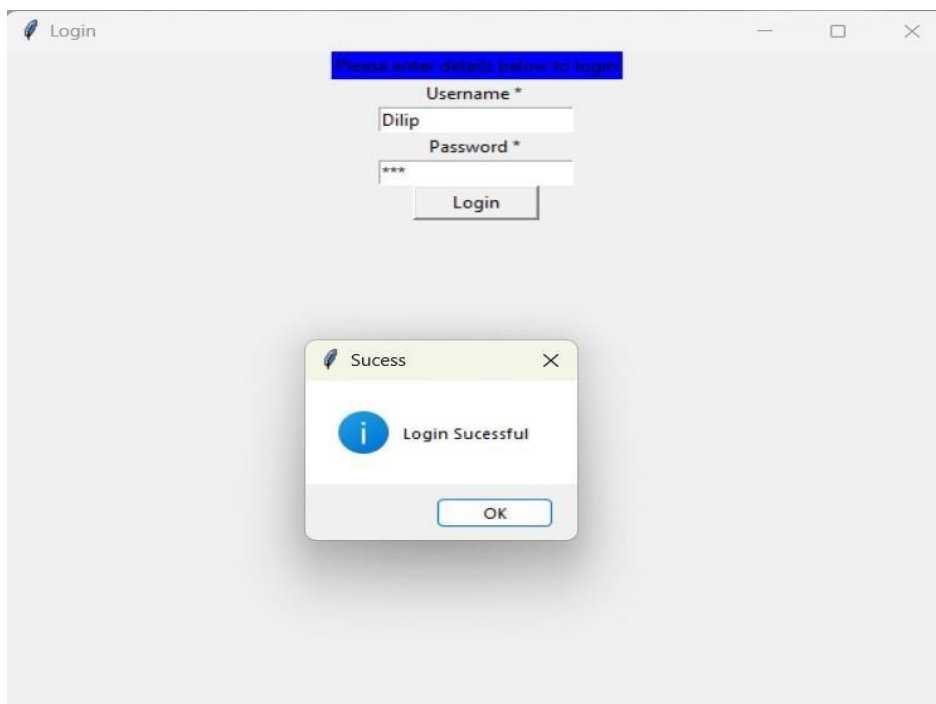
The screenshot shows a web browser window with the title 'Register'. The page has a light gray background. At the top, there is a green banner with the text 'Registration Success'. Below the banner, there is a button with the text 'Click Here to proceed'.

### 8.3. Login:



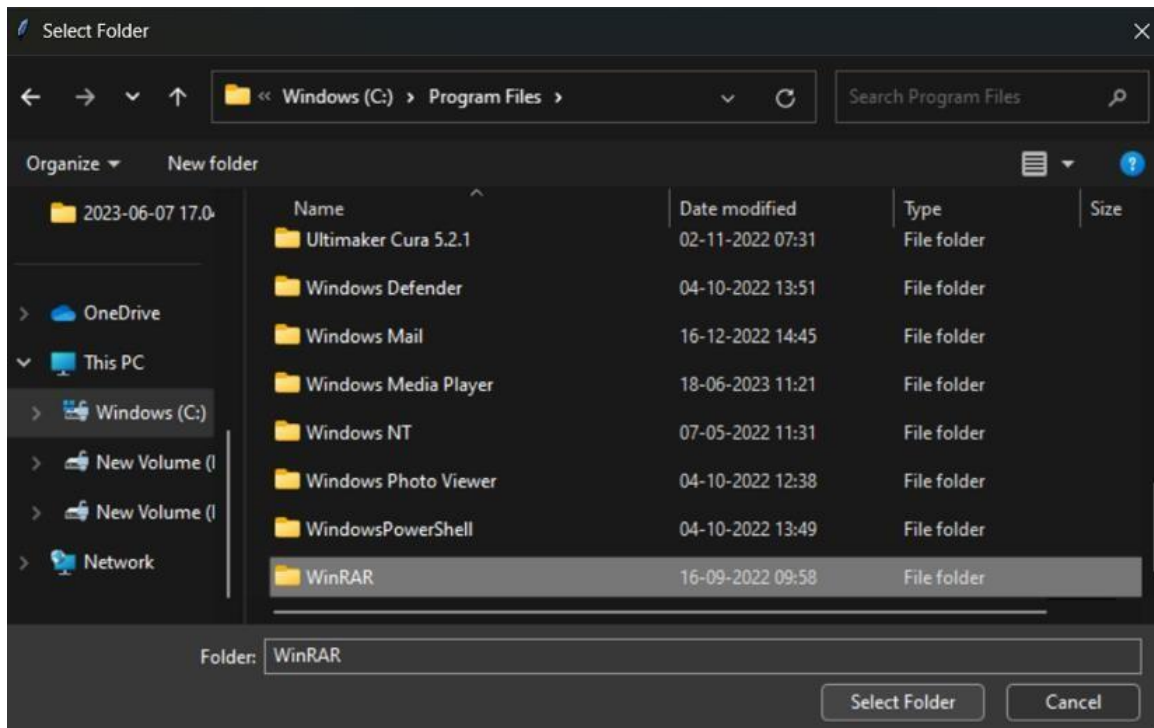
A screenshot of a 'Login' window. At the top, a blue banner contains the text 'Please enter details below to login'. Below this, there are two input fields: 'Username \*' with the text 'Dilip' and 'Password \*' with three asterisks '\*\*\*'. A 'Login' button is positioned below the password field. The window has a standard title bar with a feather icon and the text 'Login'.

### 8.4. Login Successful:

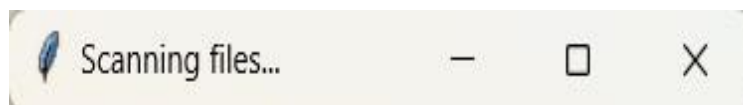


A screenshot of the 'Login' window with a 'Sucess' (sic) dialog box overlaid. The dialog box has a yellow header with a feather icon and the text 'Sucess'. It features a blue information icon and the text 'Login Successful'. An 'OK' button is at the bottom of the dialog. The background 'Login' window is still visible, showing the same fields and banner as in the previous screenshot.

## 8.5. Choose Folder:

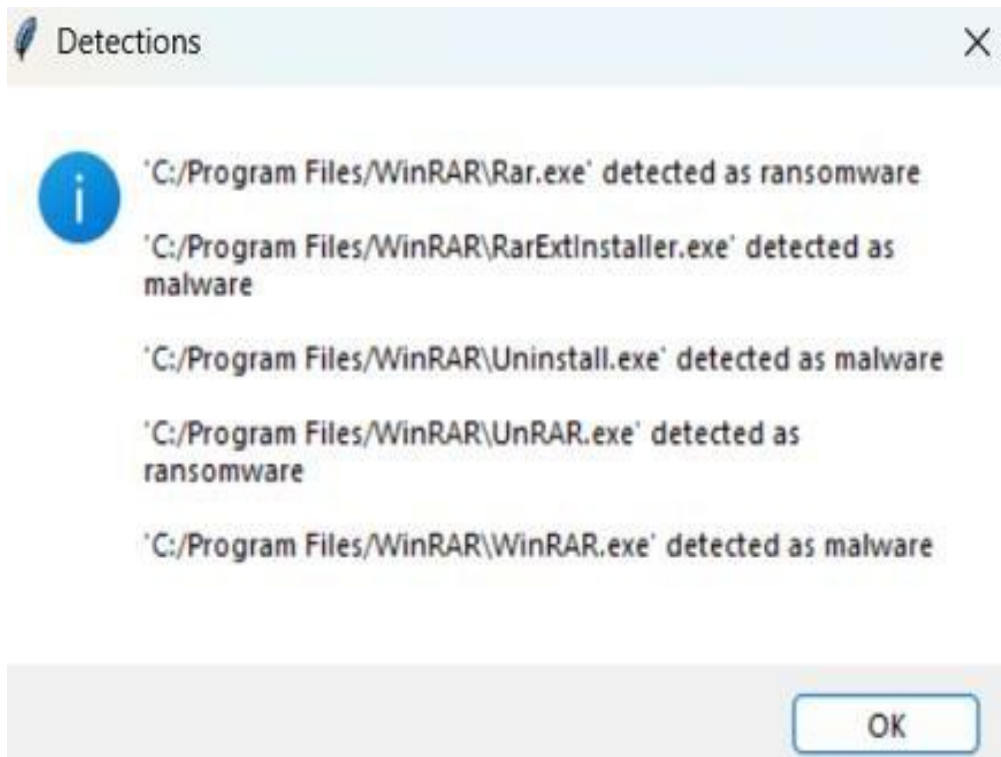


## 8.6. Scanning Files:





### 8.7. Result:



## **CHAPTER-9**

### **SYSTEM TESTING**

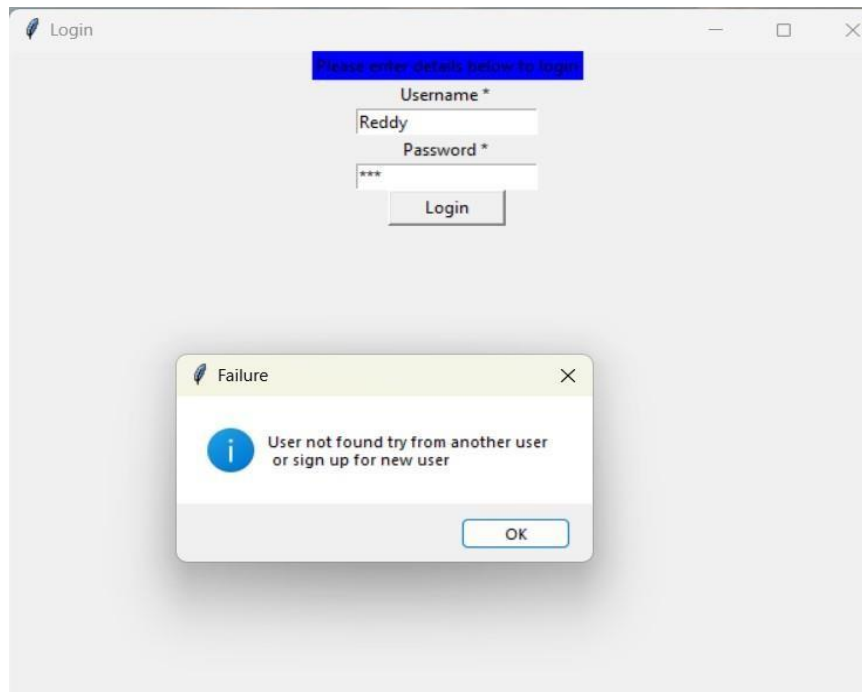
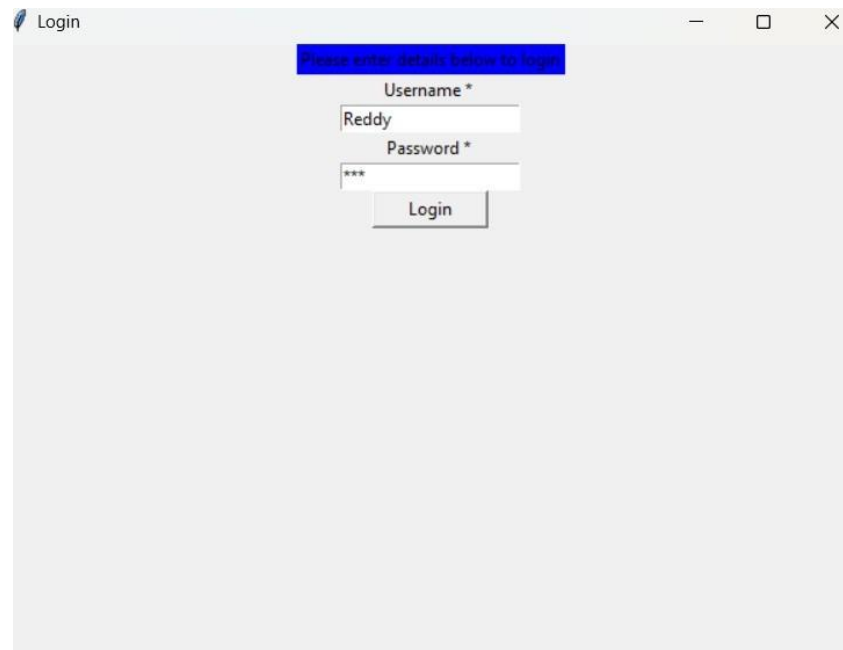
The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

#### **TYPES OF TESTS**

##### **Unit Testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.





Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

### **Test objectives**

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

### **Features to be tested**

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

## CONCLUSION

In this application, we have successfully created a system to predict whether the uploaded folder contains Ransomware or not. This is developed in a user-friendly environment using Flask via Python programming. The system is likely to collect information from the user to predict the requirements. The proposed system for ransomware detection using Artificial Neural Networks (ANN) offers a promising approach to combat the ever-growing threat of ransomware attacks. Through the integration of opcode models and grayscale models, the system leverages the power of deep learning to detect ransomware with improved accuracy and efficiency. By combining opcode sequences and grayscale representations as input data, the ANN model learns to distinguish between ransomware and benign files based on their distinct patterns and behaviors.

The opcode model enables the analysis of low-level instructions executed by files, while the grayscale model visualizes binary content, allowing for more comprehensive and effective feature extraction. The fusion of these modalities enriches the model's ability to identify ransomware in diverse forms and emerging variants.



This application can be extended to include the ability to predict multiple anti-virus and security issues. we plan to explore the prediction methodology using the updated viruses and use the most accurate and for predicting appropriate methods.

## REFERENCES

- [1] Baig, M.; Zavarsky, P.; Ruhl, R.; Lindskog, D. The study of evasion of packed PE from static detection. In Proceedings of the World Congress on Internet Security (WorldCIS), Guelph, Ontario, 10–12 June 2012; pp. 99–104.
- [2] Zakaria, W.Z.A.; Mohd, M.F.A.O.; Ariffin, A.F.M. The Rise of Ransomware. In Proceedings of the 2017 International Conference on Software and e-Business, ICSEB 2017, Hong Kong, 28–30 December 2017; pp. 66–70.
- [3] Narayanan Balkrishnan and Dr. Govindarajan Muthukumarasamy. Crop production Ensemble Machine Learning model for prediction. The 11th ACM International Conference on Management of Digital EcoSystems (2019).
- [4] Kolter, J.Z.; Maloof, M.A. Learning to detect and classify malicious executables in the wild. *J. Mach. Learn. Res.* 2006, 7, 2721–2744.
- [5] Hu, Z., Chen, X., Fu, S., Nika, A., & Xing, L. (2018). Ransomware Detection using Recurrent Neural Networks. In Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (ASIA CCS s'16).
- [6] Scaife, N., Carter, B., Traynor, P., & Butler, K. (2016). A Look at the Modern Ransomware Landscape: Research and Insights. In Proceedings of the 2016 Internet Measurement Conference (IMC '16).
- [7] Zhang, Y., Zhang, X., Cui, W., Yang, J., & Guo, Z. (2019). A Ransomware Detection Method Based on Deep Learning in Cloud Computing Environment. In 2019 IEEE International Conference on Big Data (Big Data '19).
- [8] Ramachandran, D., Feichtner, T., Hohlfeld, O., Gasser, O., & Holz, T. (2020). DeepRansom: A Generative Ransomware Detection System. In 2020 IEEE Symposium on Security and Privacy (SP '20).
- [9] Abbas, N. H., & Schiavoni, V. (2020). Ransomware Detection via Deep Learning: A Survey. *IEEE Access*, 8, 219016-219028.