

**University Institute of Information Technology,**

**PMAS-Arid Agriculture University,**

**Rawalpindi Pakistan**

**Project Name**

***By***

**Mudassar Riaz 17-ARID-1493**

**Bilal Rafique 17-ARID-1462**

**Ali Hamza 16-ARID-846**

***Supervisor*Dr./Mr./Ms. Supervisor Name**

***Bachelor of Science in Computer Science (20xx-20xx)***

***OR Information Technology OR Software Engineering***

**The candidate confirms that the work submitted is their own and appropriate  
 credit has been given where reference has been made to the work of others**.

**DECLARATION**

We hereby declare that this software, neither whole nor as a part has been copied out from any source. It is further declared that we have developed this software documentation and accompanied report entirely on the basis of our personal efforts. If any part of this project is proved to be copied out from any source or found to be reproduction of some other. We will stand by the consequences. No Portion of the work presented has been submitted of any application for any other degree or qualification of this or any other university or institute of learning.

Student Name 1 Student Name 2 Student Name 3

--------------------------- --------------------------- ---------------------------

**CERTIFICATE OF APPROVAL**

It is to certify that the final year project of BS (CS/IT/SE) “Project title” was developed by “Student **Name, Registration #”**, “Student **Name, Registration #”** and “**Student Name, Registration #”** under the supervision of “Supervisor Name” and that in their opinion; it is fully adequate, in scope and quality for the degree of Bachelors of Science in Computer Science/Information Technology/Software Engineering.

---------------------------------------

**Supervisor**

---------------------------------------

**External Examiner (If any)**

---------------------------------------

**Administrator UIIT**

**Executive Summary**

In public places, there is often a need for monitoring people and different activities going on, which can be referred later for many reasons including security. Appointing humans for this task involves many problems such as increased employee hiring, accuracy problem, trust, no proof for later use, and also the fact that a human can remember things till a certain time limit. Talking about the current security system, they use dumb still cameras with a continuous recording facility ir-respective of the fact that any event may happen or not. Moreover they are usually pointing at a specific user defined locations so more than one cameras are required to cover the entire region.

To prevent all these problems from prevailing, the CSCS is developed. It is a surveillance system, which provides solution to many of these problems. It is a stand-alone application which doesn’t require any computer to operate. It monitors different situations using a camera which is able to rotate intelligently based on sensor messages and captures the scene in the form of video or photos later reference as well.

**C**ustomizable **S**urveillance **C**ontrol **S**ystem **(CSCS)** is a surveillance system that can be assigned a sensor type as in our case a heat sensor is used, it works accordingly, rotates the camera upon event detection and perform user defined actions like capturing video and stores them, for the future use.

It is an embedded system consisting of Linux fox kit with embedded a running server application also a camera, USB storage device and a sensor node base station is attached with fox kit. LAN communication is used by user to download the videos and to operate the system manually.

**Acknowledgement**

All praise is to Almighty Allah who bestowed upon us a minute portion of His boundless knowledge by virtue of which we were able to accomplish this challenging task.

We are greatly indebted to our project supervisor “Dr. Kashif Sattar” and our Co-Supervisor “Dr. Tariq Ali” for personal supervision, advice, valuable guidance and completion of this project. We are deeply indebted to him/her/them for encouragement and continual help during this work.

And we are also thankful to our parents and family who have been a constant source of encouragement for us and brought us the values of honesty & hard work.

Student Name 1 Student Name 2 Student Name 3

--------------------------- --------------------------- ---------------------------

**Abbreviations**

|  |  |
| --- | --- |
| **SRS** | Software Requirement Specification |
| **PC** | Personal Computer |
|  |  |
|  |  |
|  |  |

**Table of Contents**

**Introduction 1**

[1.1 Brief](#_Toc268523777) 2

[1.2 Relevance to Course Modules](#_Toc268523779) 2

[1.3 Project Background](#_Toc268523780) 3

[1.4 Literature Review](#_Toc268523780) 3

[1.5 Methodology and Software Life Cycle](#_Toc268523782) 5

**Problem Definition** 6

[2.1 Purpose](#_Toc268523795) 7

[2.2 Product Functions](#_Toc268523796) 7

[2.3 Proposed Architecture](#_Toc268523804) 7

[2.4 Project Deliverables](#_Toc268523806) 8

[2.5 Operating Environment](#_Toc268523807) 8

[2.6 Assumptions and Dependencies](#_Toc268523808) 8

**Requirement Analysis** 9

[3.1 Functional Requirments](#_Toc268523823) 10

[3.2 Non – Functional Requirments](#_Toc268523825) 11

[3.2.1 Usability](#_Toc268523787) 11

[3.2.2 Reliability](#_Toc268523787) 11

[3.2.3 Performance](#_Toc268523787) 11

[3.2.4 Supportability](#_Toc268523787) 11

[3.2.5 Design Constraints](#_Toc268523787) 11

[3.2.6 Licensing Requirements](#_Toc268523787) 11

[3.3 Use case Model](#_Toc268523823) 12

[3.3.1 Use Case Diagarm](#_Toc268523787) 12

[3.3.2 Actors Discription](#_Toc268523787) 16

[3.3.3 Use Case Discription](#_Toc268523787) 17

**The Design** 21

[4.1 UML Structural Diagrams](#_Toc268523830) 22

[4.1.1 Component Diagram](#_Toc268523787) 22

[4.1.2 System Component Diagram](#_Toc268523787) 24

[4.1.3 Package Diagram](#_Toc268523787) 25

[4.1.4 Deployment Diagram](#_Toc268523787) 26

[4.2 UML Behavioral Diagrams](#_Toc268523830) 27

[4.2.1 Activity Diagrams](#_Toc268523787) 27

[4.2.2 State Machine Diagrams](#_Toc268523787) 29

[4.3 UML Interaction Diagrams](#_Toc268523830) 30

[4.3.1 Sequence Diagrams](#_Toc268523787) 30

[4.4 Node Structure](#_Toc268523830) 31

[4.5 Communication Design Protocol](#_Toc268523830) 32

**Implementation** 33

[5.1 Communication Protocol Implementation](#_Toc268523830) 34

[5.2 PC Application Implementation](#_Toc268523830) 36

[5.3 Embedded Application Implementation](#_Toc268523830) 38

[5.4 Wireless Sensor Application Implementation](#_Toc268523830) 51

**Testing and Evaluation** 52

[6.1 Verification](#_Toc268523830) 53

[6.1.1 Functional Testing](#_Toc268523787) 53

[6.1.2 Static Testing](#_Toc268523787) 57

[6.2 Validation](#_Toc268523830) 57

[6.3 Usability Testing](#_Toc268523830) 57

[6.4 Unit Testing](#_Toc268523830) 57

* 1. [Integration Testing](#_Toc268523830) 57

6.6 [System Testing](#_Toc268523830) 57

**GUI** 58

**Future Work** 70

**References** 72

**List of Figures**

Fig 1.1 Block Diagram 8

Fig 2.1 Use Case Diagram 9

# Chapter 1: Introduction

This chapter provides the overview of the project. The first paragraph of every chapter should provide the chapter summary.

# Brief

The aim of this project is to help cyber security experts to understand the behavior of malware based on Api’s. In this project we have performed classification of malware families according to their behavior using Api calls.

# Relevance to Course Modules

A brief explanation of how your project is related to various courses studied during degree.

# Project Background

The struggle between security analysts and malware developers is a never-ending battle with the complexity of malware changing as quickly as innovation grows. Current state-of-the-art research focus on the development and application of machine learning techniques for malware detection due to its ability to keep pace with malware evolution. This survey aims at providing a systematic and detailed overview of machine learning techniques for malware detection and in particular, deep learning techniques..

# Literature Review

[1](Gibert, Mateu, & Planes, 2010)The struggle between security analysts and experts and malware developers is a never ending battle with the complications of malware changing as quickly as innovation grows. Current state-of-the-art research focus on the development and application of machine learning techniques for malware detection due to its ability to keep pace with malware evolution. This survey aims at providing an organized and detailed overview of machine learning techniques for malware detection and in particular deep learning techniques. It provides absolute explanation of the methods and features in a traditional machine learning work ﬂow for malware detection and classiﬁcation it explores the challenges and limitations of traditional machine learning and it analyzes recent trends and developments in the ﬁeld with special importance on deep learning approaches.

[2](M. Siddiqui et al, 2009) Used Data Mining for detection of Worms. They used variable length instruction sequence. Their Primary data set consists of 2,775 Windows PE files, in which in which 1,444 were worms and 1,330 were benign. They performed detection of compilers, common packers and crypto before disassembly of files. Sequence reduction was performed and 97% of the sequences were removed. They used Decision Tree, Bagging and Random Forest models using. Random forest performed slightly better than the others.

[3](Shabtai et al, 2009)Provide a taxonomy for malware detection using machine learning algorithms by reporting some feature types and feature selection techniques used in the literature. They mainly focus on the feature selection techniques(Gain ratio, Fishers core, document frequency, and hierarchical features election) and classiﬁcation algorithms (Artiﬁcial Neural Networks, Bayesian Networks, Natïve Bayes, K-Nearest Neighbor, etc). In addition, they review how ensemble algorithms can be used to combine a set of classiﬁers

[4](J. Z. Kolter et al, 2004.) Used n-gram analysis and data mining approaches to detect malicious executables in the wild. The authors used a hexdump utility to convert each executable to hexadecimal code in an ASCII format and produced n-gram features by combining each four-byte sequence into a single term. Their primary dataset consisted of 1971 clean and 1651 malicious programs They used different classifiers including Instance-based Learner, TFIDF, Naive-Bayes, Support vector machines, Decision tree, boosted Naive-Bayes, SVMs and boosted decision tree. They used information gain to select valued features which are provided as input to all classifiers. The area under an ROC curve (AUC) is a more complete measure compared with the detection accuracy as they reported that the boosted decision trees outperform rest of the classifiers for both classification problems.

# Analysis from Literature Review (in the context of your project)

This section provides an analytical discussion of your work in comparison with discussion in literature review.

# Methodology and Software Lifecycle for This Project

There are different types of methodologies are used to building a software or any of the project. We have studied all the types of methodologies that can be used but from all of them we select the method that best fit to our project is “Extreme Programming”.

We are selecting this model because it is:

Best suited for: Projects that require maintaining stringent stages and deadlines,or projects that have been done various times over where chances of surprises during the development process are relatively high.

One more reason is that this method is applied where the requirements are not very much clear. So that will happen with our project too so that’s why we are selecting this Method.

In systems design, and particularly software design, a common methodology for the development of a new system is the Systems Development Life Cycle, or SDLC. The SDLC contains the following phases of systems development:

• **Planning**

Determine the purpose of the system.

• **Analysis**

Determine what the system needs to do, the goals for the system and how to determine if those goals have been met.

• **Design**

Determine how the system will work, what the overall architecture is, and determine what steps would need to be taken to construct an actual system.

• **Implementation**

Using the existing design, we will construct a system to meet the requirements of the project.

• **Testing**

Establish that the constructed system actually does meet the requirements detailed in the design.

• **Maintenance**

Fix bugs in the system, which are essentially differences between the design (requirements) and the constructed system (reality). As the design inevitably changes, update the actual system to match these changes.

**Chapter 2: Problem Definition**

This chapter discusses the precise problem to be solved. It should extend to include the outcome.

# Problem Statement

The Malware threats has been with us since the dawn of computing. The earliest documented virus appeared during the 1970s. It was known as the Creeper Worm and was an experimental self-replicating program that copied itself to remote systems and displayed the message: “I’m the creeper, catch me if you can”. Later, in the early 80s, appeared Elk Cloner, a boot-sector virus that targeted Apply II computers. From these simple beginnings, a massive industry was born and, since then, the fight against malware has never stopped.

# Deliverables and Development Requirements

We will deliver the project step by step as guided by our supervisor. The most important things will deliver first and then gradually build the whole System.

# Proposed Architecture

My purpose of the project to provide the security to the systems from malware. Security is the thing that every system requires that wants to communicate over the internet. Many of the malicious codes or files attached to your files when you are going to download anything from the internet. Different type of malware are present on the internet world, every specie has its own purpose and method that how to attack on any computer. So that’s why we are going to build that type of model that can classify any type of malware.

**2.4 Product Functions**

1. Malware Profiling of Reports(Basic Structure,Low level behaviour, High Level behaviour)
2. Feature Generation of on the basis on malware profiling
3. Detection
4. Classification
5. Evaluation

**2.5 Operating Environment**

One cuckoo host. One switch that control traffic between host and guest there three Guest.The host

manage guest, start analysis and get reports. Ubuntu as a host operating system and window 10, window

7 as a guest are installed in virtual box and cuckoo agent is used to work as communication medium

in guest and host.

# Chapter 3: Requirement Analysis

Software Requirements Specification (SRS) report should be included in this chapter.

# Use Cases

Use cases are a widely used and highly regarded format for capturing requirements. Before writing functional requirement use cases can help you to understand the requirements in the way user expect. Following table presents you not only the template to write use case(s) as well as guides you to write each section with example.

|  |  |
| --- | --- |
| Use case ID | ARID-846 |
| Use case Name | Report Generation |
| Actor Name | Direct User |
| Description | By selecting the files from the system after processing it will generate report about selected data or files |
| Trigger | To accomplish this task we Click on button (Select File) |
| Precondition | NO info about Dataset |
| Post Condition | Dataset provide complete information on the basis on Malware |
| Normal Flow | 1.Chose the dataset from system  2.select dataset and upload it for action  3.send dataset to sandbox  4.Then select action report generate |
| Alternative Flow | If success fully then move to the next process else request user to try again |
| Expectation | During taking action some errors can be occurs  1.No file selected  2.Folder having no dataset  3.Uploading Error  In such type of condition user request to restart from initial stage. |
| Includes: | null |

|  |  |
| --- | --- |
| Use case ID: | ARID-846 |
| Use case Name | Data Collection |
| Actor Name | Direct User |
| Description: | For collection of data first we take report &Api’s,Database when all these steps completed then process start for Data Collection |
| Trigger | For the process this task first get Report and Api’s,Database |
| Precondition | Report in not in data collection |
| Post Condition | Report in Database |
| Normal Flow: | 1.Input report get from sandbox  2.Save report into Data Collection  3.Check Successful Save or not |
| Alternative Flow: | If save successfully then move to the next step else again request save. |
| Exception: | During the performing action some error can be take place  1.Report not be selected from Sandbox  2.report not saved |
| Includes: | null |

|  |  |
| --- | --- |
| Use case ID: | ARID-846 |
| Use case Name | Profiling |
| Actor Name | Direct User |
| Description: | For profiling following steps should take place  1. Start from system Action.  2. Get report from Data Collection.  3. Perform Profiling.  4.Successfully perform or fail. |
| Trigger: | To complete this task take report from Data Collection |
| Precondition | No profile info |
| Post condition | Profiling to their prospective behavior |
| Normal Flow | 1. Get report from Data Collection.  2. Perform Profiling.  3. Successfully perform or fail |
| Alternation | Successfully move next else request for data set  From dataset |
| Exception: | During the performing action some error can be take place  1.Report not be selected from Data Collection  2.report not saved  3.Fail to Profiling |
| Includes: | Null |

|  |  |
| --- | --- |
| Use Case ID: | ARID-846 |
| Use Case Name | Feature Generation |
| Actor Name: | Direct User |
| Description: | System Action:  1)Get Profiling detail  2)Perform Feature Generation  3)Success |
| Trigger | Get Profile Data() from Api,DLL,Network,File |
| Precondition | The data was only Profiled |
| Post condition | Feature generated from profiled |
| Normal Flow | For performing this process there are following steps  1)Get Profiling detail  2)Perform Feature Generation  3)Success  4)Feature are generated. |
| Alternative Flow | Success full then move to next step else Move again to feature generation |
| Exception: | During the performing action some error can be take place  1.Profiling detail not collected  2.Fail to Perform feature generation |
| Includes | Null |

|  |  |
| --- | --- |
| Use case ID: | ARID-846 |
| Use case Name | Feature Database |
| Actor Name | Direct User |
| Description: | For process this task there are following steps:  1)Feature generation Report  2)save into Database  3) Successfully save or fail. |
| Trigger | Get Featured \_data From Feature Generation () |
| Precondition | No data in feature data |
| Post condition | Feature data came |
| Normal Flow | 1)System Action  2)Feature generation Report  3)save into Database  4) Successfully save or fail. |
| Alternative | If report get and save success fully then move next else Get report and save |
| Exception | Feature generation Report not selected  Report in found in database  Save to fail |
| Includes | null |

|  |  |
| --- | --- |
| Use case ID: | ARDI-846 |
| Use case Name | Detection |
| Actor Name | Direct User |
| Description | For Detection of Malware  System Action  1)Get Feature generated report database  2)detect the malware from report  3)Success |
| Trigger | Collect feature generation data from database |
| Precondition | Non Detected |
| Post condition, | Detected |
| Normal Flow | System Action  1)Get Feature generated report database  2)detect the malware from report  3)Success |
| Alternative | If successfully detected then move next step  Else again get file from database |
| Exception | Data collection fail from Feature generation report  Detection of malware fail |
| Includes | null |

|  |  |
| --- | --- |
| Use case ID: | ARID-846 |
| Use case Name | Classification |
| Actor Name | Direct user |
| Description | For Classification of Malware  System Action  1)Get data from detection  2) classify the detected data in families  3)success |
| Trigger | Detected malware file from Detection() |
| Precondition | Non Classified dataset |
| Post Condition | Classified |
| Normal Flow | 1)Get data from detection  2) Classify the detected data in families  3)Classified data set  Compelet detail of malware and their classes |
| Alternative | If success then movie to the final steps else going back to detection |
| Exception | data not collected from detection  fail to Classified data set |
| Includes | null |

**3.2 Functional Requirements:**

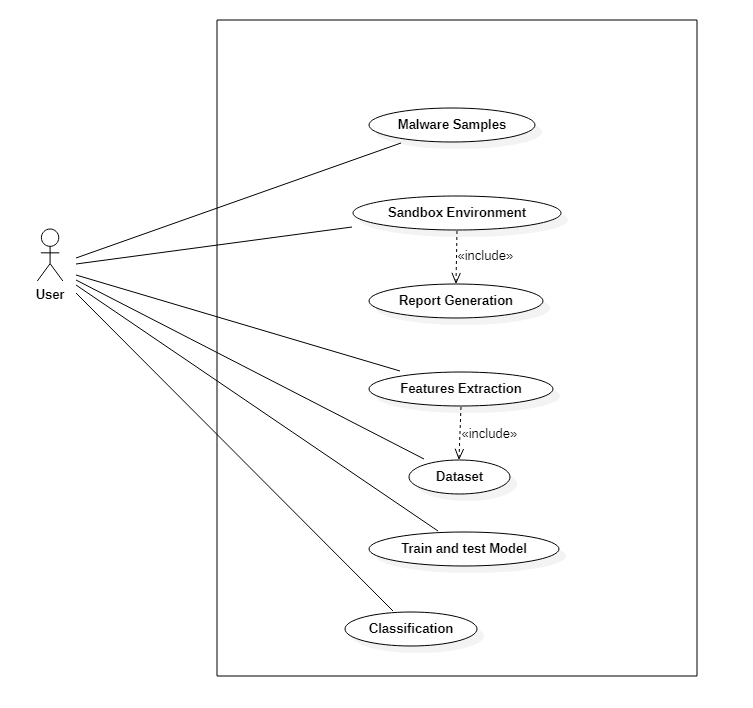
|  |  |
| --- | --- |
| **Functional Requirement No.** | **Functional Requirement Description** |
| Req.1 | User Collect Malware Samples. |
| Req.2 | Submit Malware Samples in sandbox environment. |
| Req.3 | Get generated reports from sandbox. |
| Req.4 | Extract features from these reports and prepare dataset |
| Req.5 | Now test and train dataset. |
| Req.6 | After testing and training prepare model for classification |

**3.3 Non-Functional Requirements**

|  |  |
| --- | --- |
| **Non-Functional Requirement No.** | **Non-Functional Requirement Description** |
| NFR1 | System will take sandbox generated malware reports from user as input and process it. |
| NFR2 | System will extract features from malware reports and prepare csv file . |
| NFR3 | System will provide interface by using tkinter. |
| NFR4 | MD5 will be used in for prediction. |

1. **Usability:** System should be easy to extend. The code should be written in a way that it favors implementation of new functions. It will provide the up to date information with good performance to satisfy user needs.
2. **Reliability:** This app should provide appropriate answers to the user. This app should be able to interact efficiently with the user.
3. **Integrity:** This desktop application will requires specific android version to run. It also requires an active internet connection to work and to exchange queries to provide information to the user.

**3.4 Use Case Diagram**

****

**Use case Description:**

Report generate:

* User (user Click on Button)
* System Action
* Get Dataset from System
* Send dataset to cuckoo
* Generate Report
* success fully Generated or failure

**Data collection:**

* Get Report from Sandbox
* And save that report into Data Collection
* Successful save or not

**Profiling:**

* Get Report from Data Collection
* Perform Profiling
* Successfully perform or fail

**Feature generate:**

* Feature generation Report
* Save into Database
* Successfully save or fail

**Feature database:**

* Get Feature generated report database
* Detect the malware from report
* Success and fail

**Detection:**

* Get data from detection
* Classified the detected data in families
* Success or fail

**Classification:**

* System Action
* Get data from detection
* Classified the detected data in families
* Success or fail

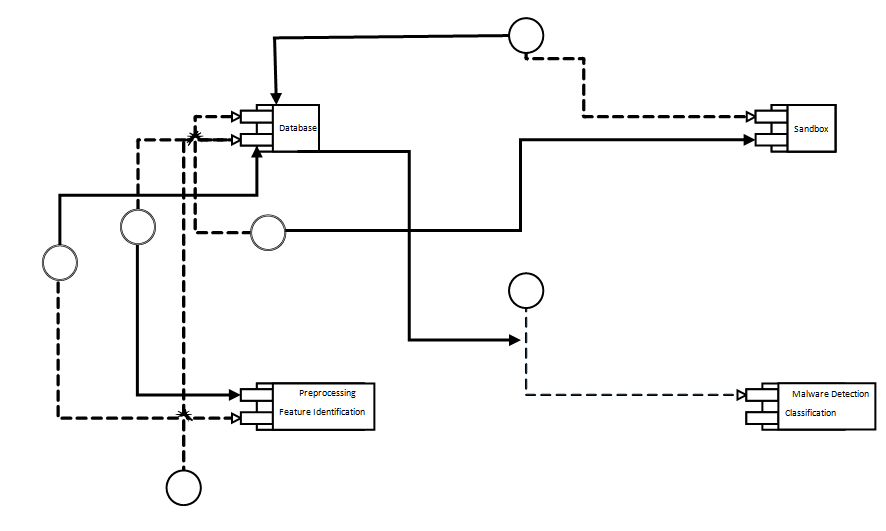
**Evaluation:**

* System Action
* Get the file from classification
* And then evaluate that
* Success send file to the user and fail

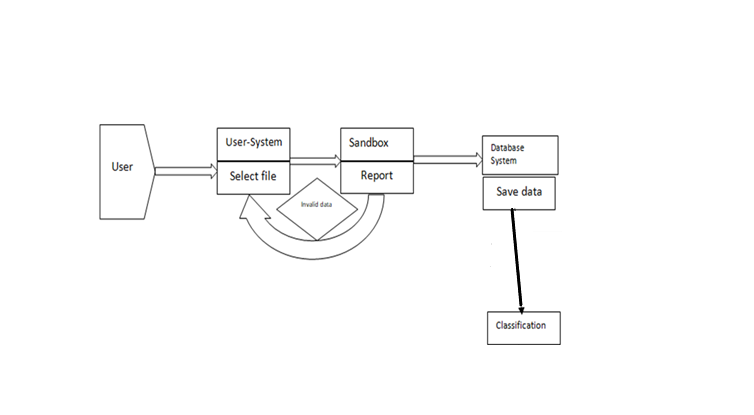
Chapter 4: Design and Architecture

**4.1 UML Structural Diagrams**

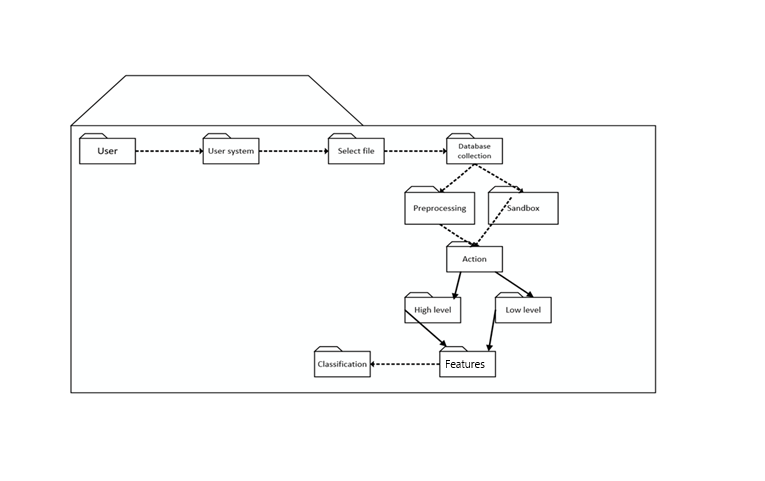
[4.1.1 Component Diagram](file:///C:\Users\MudassirRiaz\Downloads\FYP%20Final%20Report.docx#_Toc268523787)



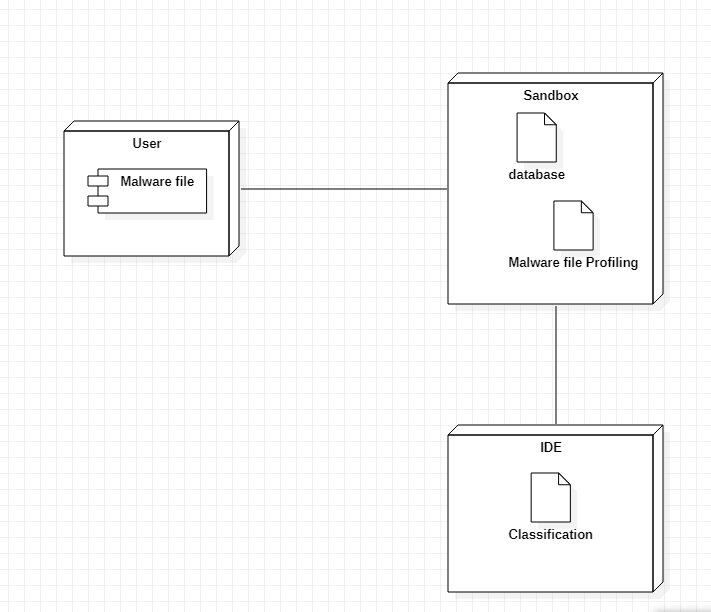
[**4.1.2 System Component Diagram**](file:///C:\Users\MudassirRiaz\Downloads\FYP%20Final%20Report.docx#_Toc268523787)

****

[**4.1.3 Package Diagram**](file:///C:\Users\MudassirRiaz\Downloads\FYP%20Final%20Report.docx#_Toc268523787)

****

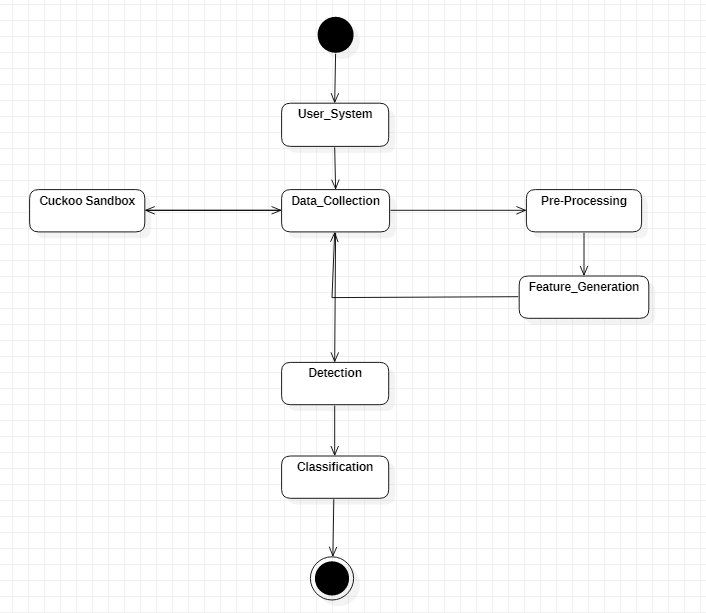
[**4.1.4 Deployment Diagram**](file:///C:\Users\MudassirRiaz\Downloads\FYP%20Final%20Report.docx#_Toc268523787)

****

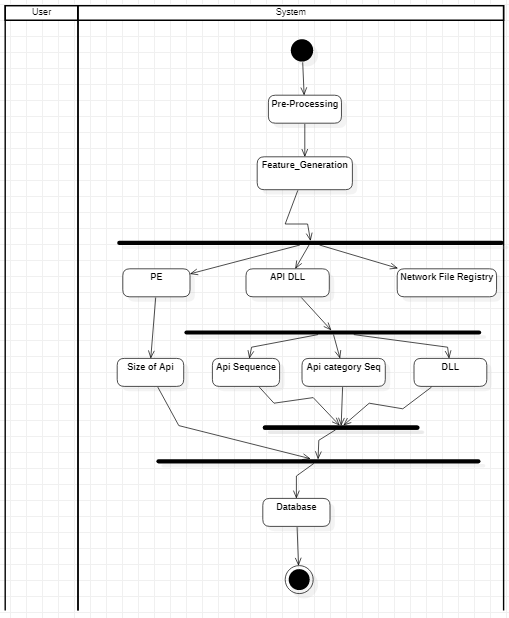
[**4.2 UML Behavioral Diagrams**](file:///C:\Users\MudassirRiaz\Downloads\FYP%20Final%20Report.docx#_Toc268523830)

[**4.2.1 Activity Diagrams**](file:///C:\Users\MudassirRiaz\Downloads\FYP%20Final%20Report.docx#_Toc268523787)

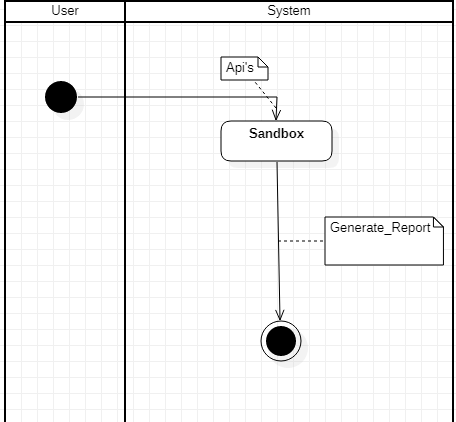
**Data Collection**



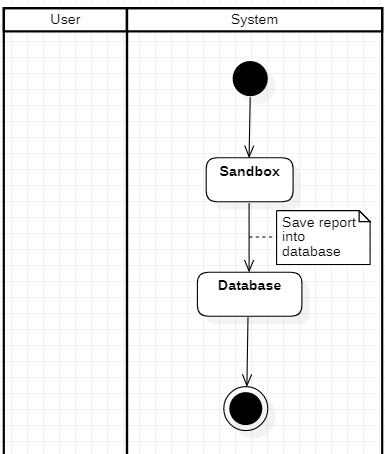
**Feature\_Extraction**



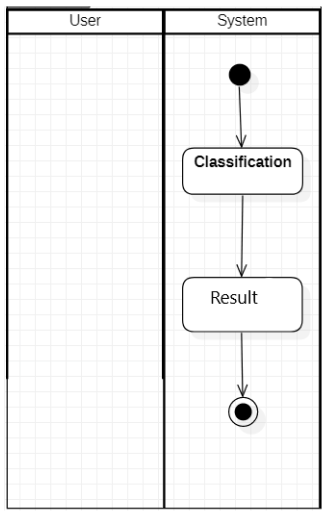
**Report\_Generation**



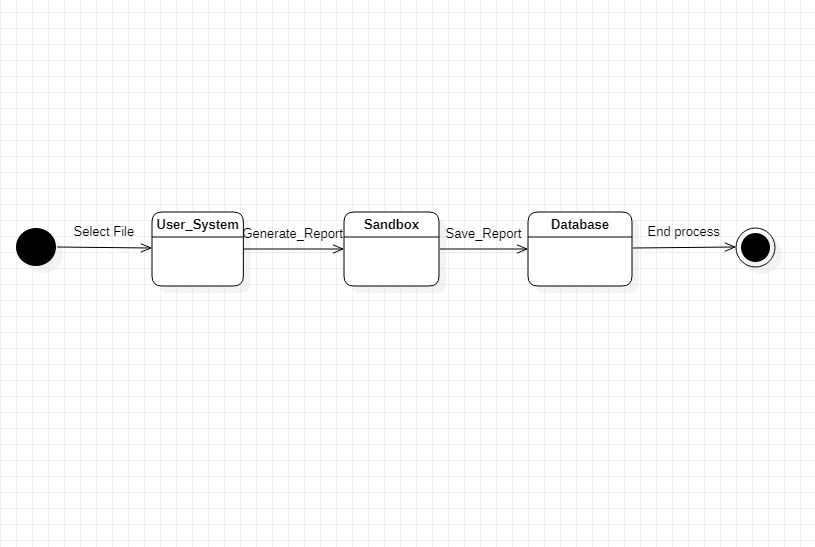
**Save Report into database**



**Classification**

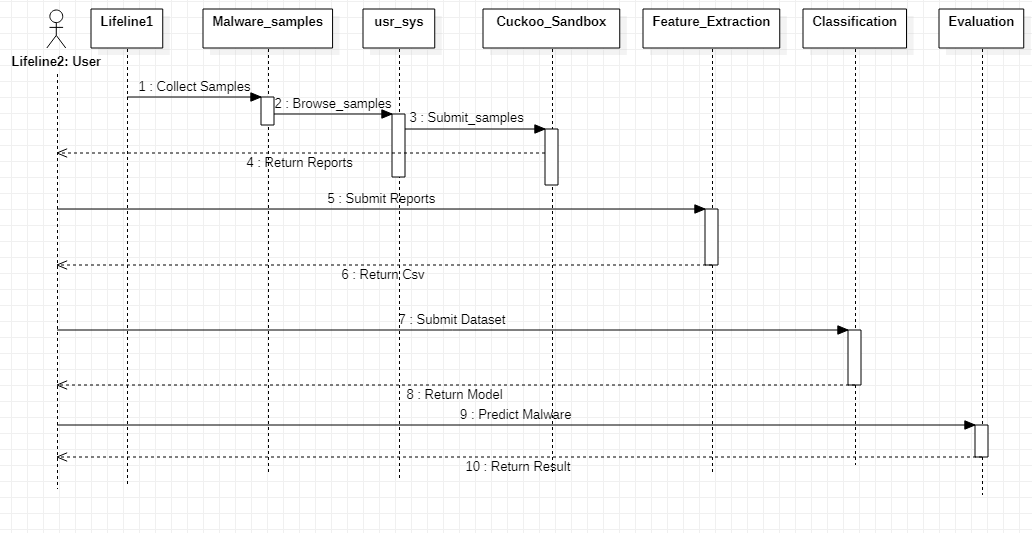
****

[**4.2.2 State Machine Diagrams**](file:///C:\Users\MudassirRiaz\Downloads\FYP%20Final%20Report.docx#_Toc268523787)

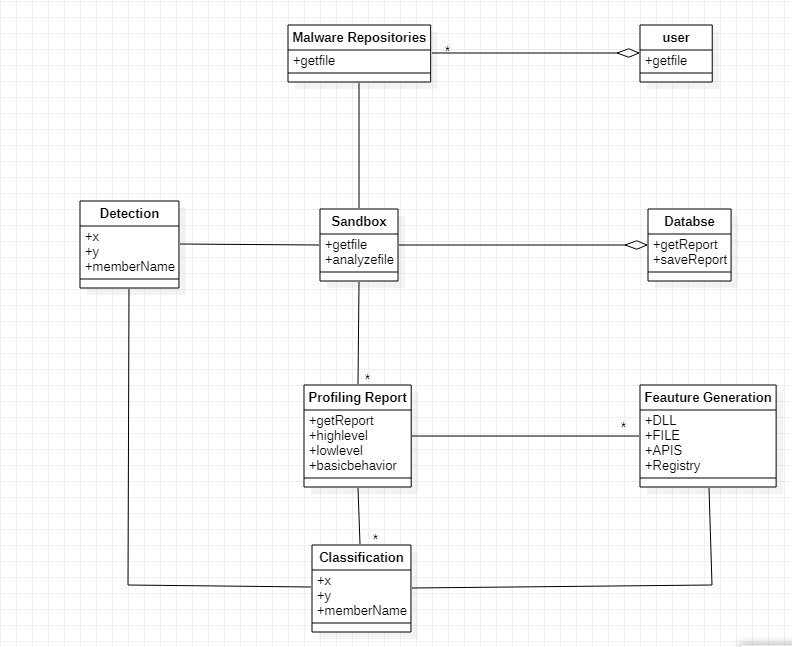


[**4.3 UML Interaction Diagrams**](file:///C:\Users\MudassirRiaz\Downloads\FYP%20Final%20Report.docx#_Toc268523830)

4.3.1 Sequence Diagram



4.3.2 Class Diagram:-



Chapter 5: Implementation

This chapter will discuss implementation details supported by UML diagrams (if applicable). You will not put your source code here. Any of the following sections may be included based on your project.

# Component Diagram

Present and explain component diagrams of your project.

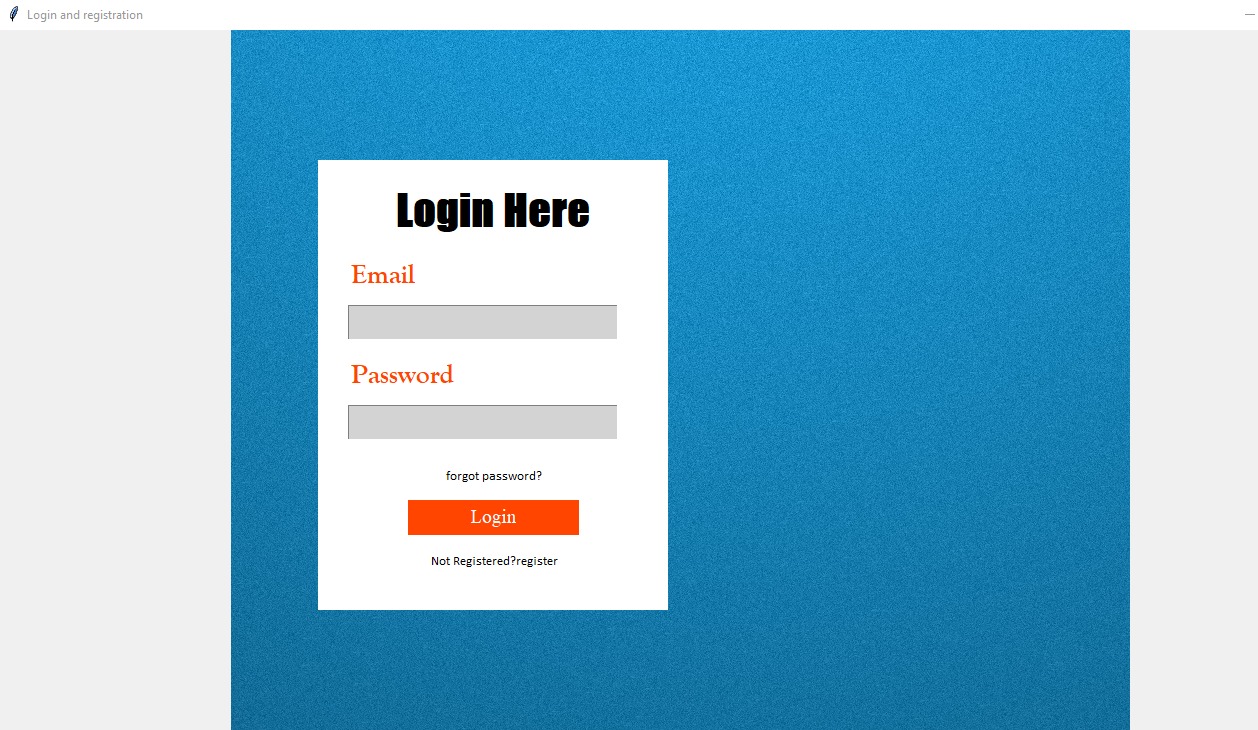
# Network and Protocol Choice

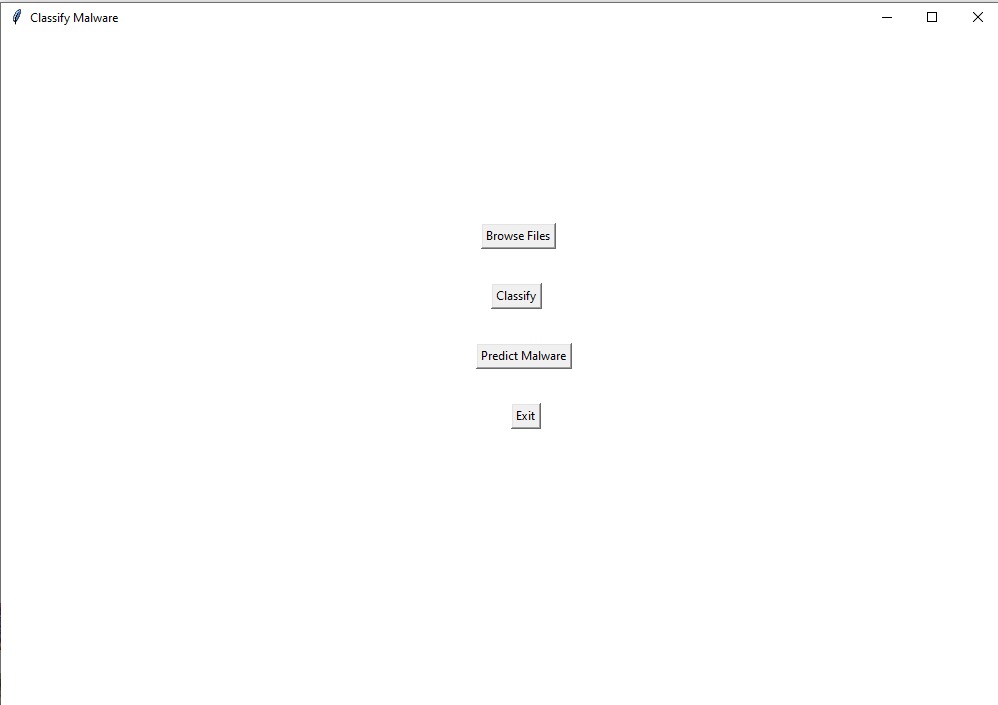
* **Design:** The designing of this malware classifier and the diagram have been done through Visio and StarUML.
* **MySQL:** It is a database system.
* Here the libraries of pycharm and tensorflow are mentioned that have been used in our project
* **Tkinter:** Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit,[1] and is Python's de facto standard GUI.
* **Tensorflow:** TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. Tensorflow is a symbolic math library based on dataflow and differentiable programming.

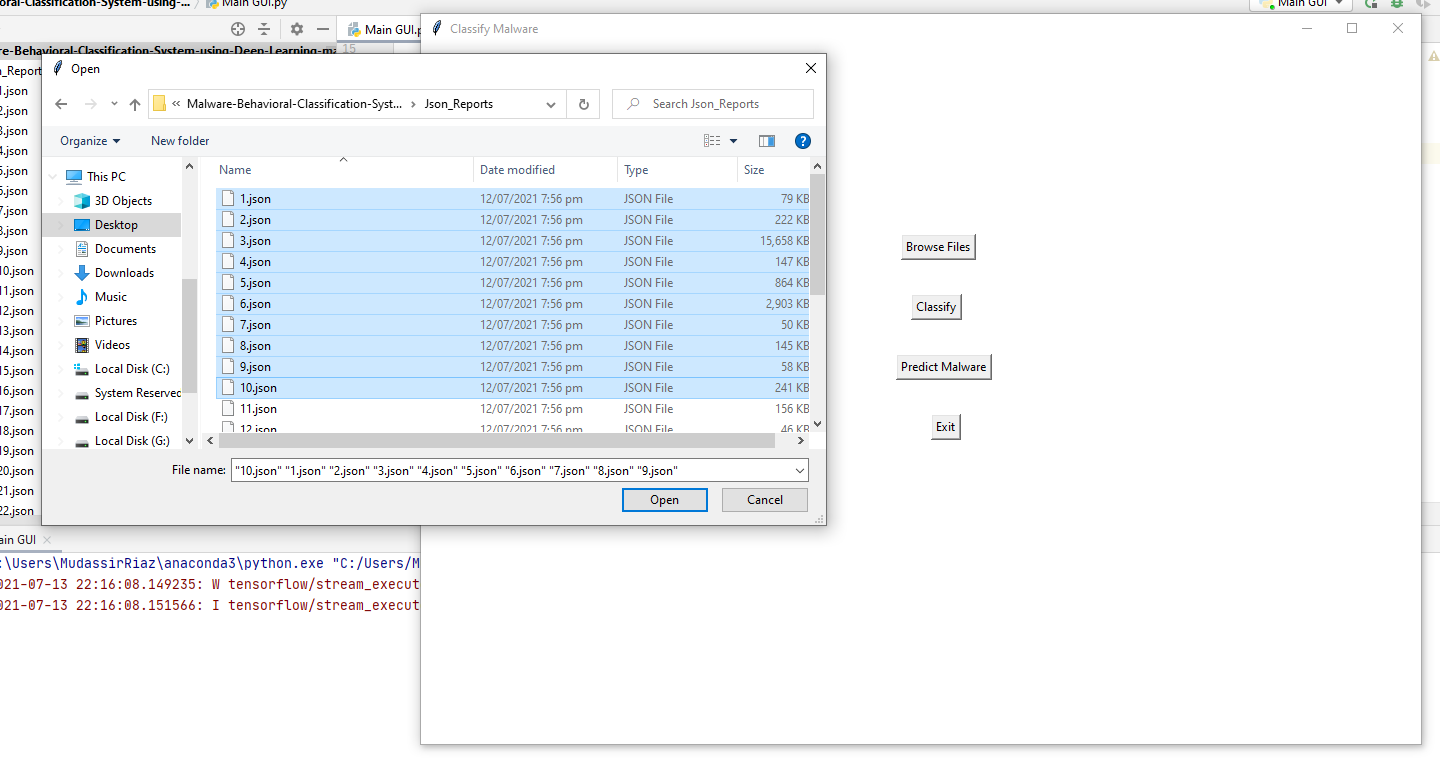
# Choice of Object Middleware

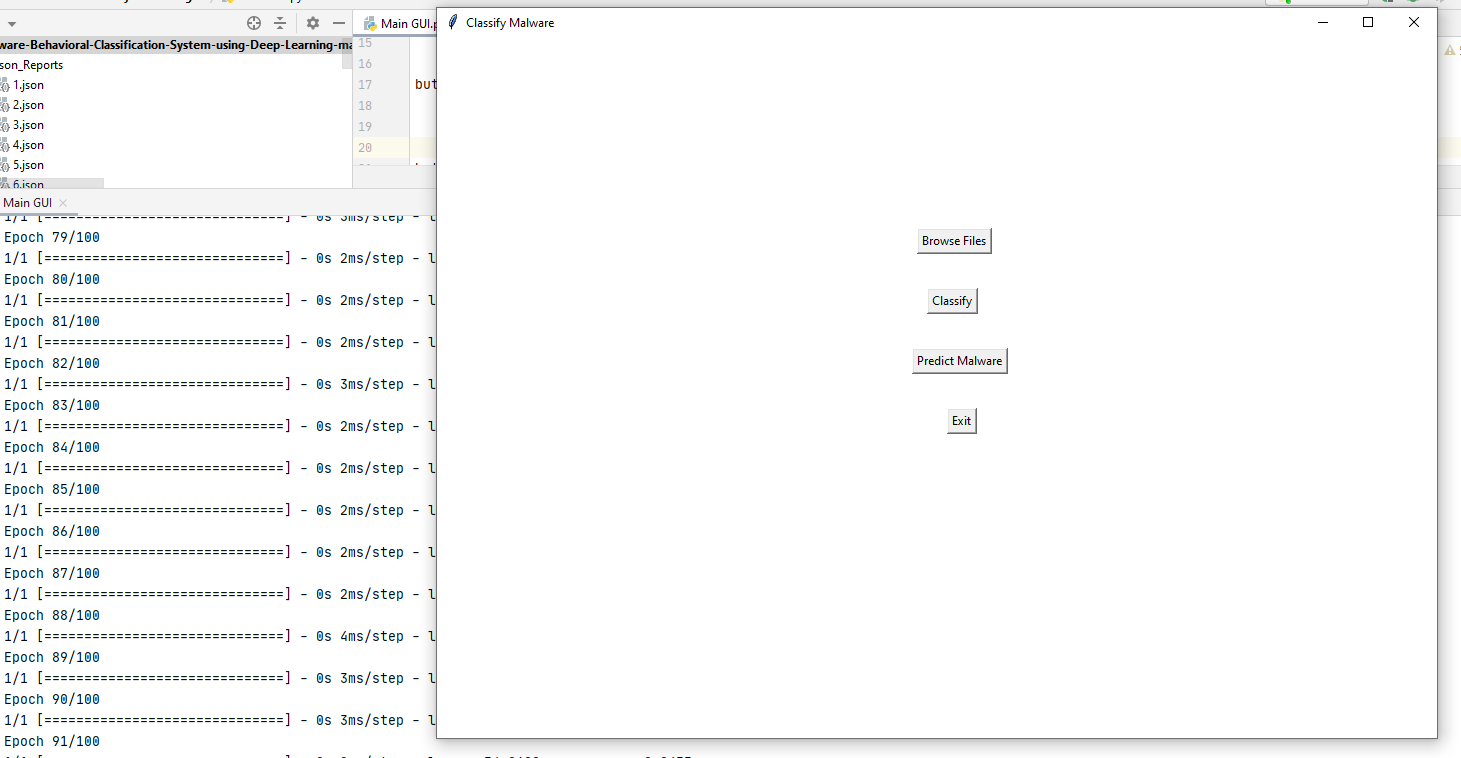
RMI vs. CORBA vs. DCOM etc.

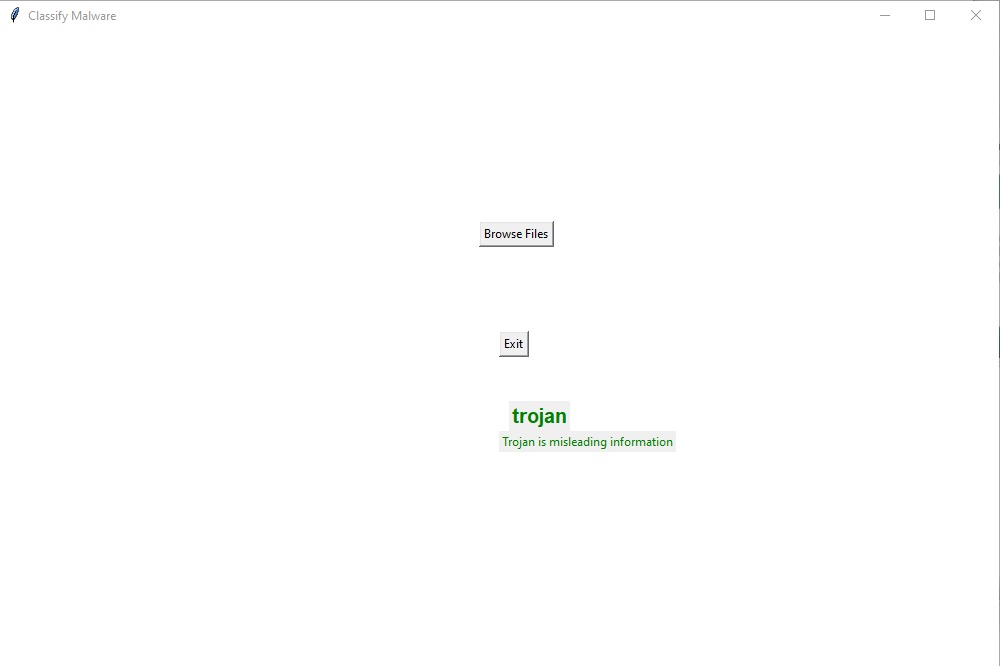
# User Interface











# Chapter 6: Testing and Evaluation

Testing is a process of executing a program or application with the intent of finding the software bugs. It can also be stated as the process of validating and verifying that a software program or application or product: Meets the business and technical requirements that guided its design and development.

# Verification

# Validation

From the customer requirement perspective, the study of design code of different software modules and also from Graphic user interfaces through which the interacts with the system. Test-Case specifications in performed for system testing by keeping in mind several issues, which are discuss in following subtopics.

# Usability Testing

# Module / Unit Testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | | | | SBC\_1 | | |
| Test priority(low,medium,high) | | | | High | | |
| Module Name | | | | Check Cuckoo conf | | |
| Description | | | | Admin Run command of cuckoo to check cuckoo is configure properly | | |
| Test Title | | | | Test configuration | | |
| Precondition | | | | Cuckoo cofig or not | | |
| Despondency | | | | Mongodb started  VM in restore mode | | |
| s. | Test steps | Expected result | Actual result | | Status fail or pass | Note | |
| 1 | * Open terminal * Run command * Cuckoo -d | At the end of console  The message will appear  “waiting for analysis” | Waiting for analysis | | Pass | If cuckoo not configure it will give message cuckoo could not find client | |

Test Case 2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Case ID | | | | SBC\_2 | | |
| Test priority(low,medium,high) | | | | High | | |
| Module Name | | | | Check Cuckoo Result | | |
| Description | | | | Admin Run cuckoo console and cuckoo web | | |
| Test Title | | | | Test result | | |
| Precondition | | | | Cuckoo produce result or not | | |
| Despondency | | | | Mongodb started  VM in restore mode  Cuckoo web | | |
| s. | Test steps | Expected result | Actual result | | Status fail or pass | Note | |
| 1 | * Open terminal * Run command * Cuckoo –d * Cuckoo web * Open browser * Type localhost:8000 * Submit upload file * Submit file | At the end of cuckoo console show message  “Analysis is completed”  And browser it will show All detail  If score is 0.0 this means cuckoo is not configure properly  It the score between  1-10  This mean cuckoo is producing accurate result | Score 3.5 | | Pass | If score is 0.0 it means we have to review cuckoo configuration | |

**Test Case 3**

Malware Classification

|  |  |
| --- | --- |
| Test Case ID | WI\_6 |
| Test priority(low,medium,high) | High |
| Module Name | Classification |
| Description | After uploading file and completion of analysis it will show of malware classification. |
| Test Title | Classification |
| Precondition | Unclassified |
| Despondency | Running cuckoo  Mongo dB should start  Virtual machine starts or restore mode  Should complete analysis |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| s. | Test steps | Expected result | Actual result | Status fail or pass | Note |
| 1 | * Open GUI * In cuckoo module * Browse file * Upload file | Predict Malware class | It show message this a trojan horse | Pass |  |

# Integration Testing

Integration testing.

# System Testing

All the modules will be tested for their properly functionality, to see the efficient working of the whole software.

# Acceptance Testing

Acceptance testing.

# Stress Testing

The software would be tested with large datasets to see if it would be working effectively under large sums of data.

# Hardware Configuration for Testing

The preferred hardware for our system is minimum Core i3 4th generation:

* Processor: 1 gigahertz (GHz) or faster
* RAM: 2 gigabyte GB (64-bit) or more
* Hard disk space: 20 GB (64-bit) or more

# Chapter 7: Conclusion and Future Work

This chapter concludes the project and highlights future work.

# Conclusion

Conclusion section.

# Future Work

Future work section.

# References

References to any book, journal paper or website should properly be acknowledged. Please consistently follow the style. The following are few examples of different resources i.e. journal article, book, and website.

1 Lyda M.S. Lau, Jayne Curson, Richard Drew, Peter Dew and Christine Leigh, (1999), Use Of VSP Resource Rooms to Support Group Work in a Learning Environment, ACM 99, pp-2. (Journal paper example)

2 Hideyuki Nakanishi, Chikara Yoshida, Toshikazu Nishmora and TuruIshada, (1996), FreeWalk: Supporting Casual Meetings in a Network, pp 308-314 (paper on web) http://www.acm.org/pubs/articles/proceedings/cscw/240080/p308-nakanishi.pdf

3 Ali Behforooz& Frederick J.Hudson, (1996), Software Engineering Fundamentals, Oxford University Press. Chapter 8, pp255-235. (book reference example)

4 Page Author, Page Title, http://www.bt.com/bttj/archive.htm, Last date accessed. (web site)