



**University Institute of Information Technology,  
PMAS-Arid Agriculture University,  
Rawalpindi**

## **API Based Intelligent Malware Detection**

*By*

**Anjum Shehzad      18-Arid-2609**

**Haris Aamir      18-Arid-2643**

**Zohaib Ali Hassan      18-Arid-2735**

*Supervisor*

**Mr. Zeeshan Javed**

***Bachelor of Science in Computer Science (2018-2022)***

# 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.

Anjum Shehzad

Haris Aamir

Zohaib Ali Hassan

-----

-----

-----

# CERTIFICATE OF APPROVAL

It is to certify that the final year project of BS (CS) “**API Based Intelligent Malware Identification**” was developed by “**Anjum Shehzad (18-Arid-2609)**”, “**Haris Aamir (18-Arid-2643)**” and “**Zohaib Ali Hassan(18-Arid-2735)**” under the supervision of “**Mr. Zeeshan Javed**” and that in their opinion; it is fully adequate, in scope and quality for the degree of Bachelors of Science in Computer Science.

-----  
**(Mr. Zeeshan Javed)**  
**Supervisor**

-----  
**(Mr. Muhammad Danish)**  
**External Examiner**

-----  
**(Prof. Dr. Yaser Hafeez)**  
**Director UIIT**

# Executive Summary

We aim to present the functionality and accuracy of different machine learning algorithms to detect whether an executable is infected or clean. The first chapter will present a description of the phenomenon of Malware, software programs or pieces of code that aim to hijack computer systems to steal information or to destroy it. We will dive deeper into this topic in order to have some understanding of these malicious programs. After a brief introduction to this phenomenon, we will present the evolution of malware over time. The following is the presentation of the different protection techniques. The second chapter will introduce the field of Artificial intelligence and its benefits. We will further discuss the importance of artificial intelligence in addressing this situation. The algorithms used by us will be described and their benefits will be presented. Artificial intelligence is widely used in this field by antivirus and anti-malware programs as well as by these malicious programs, for example Polymorphic Malware uses artificial intelligence algorithms to encrypt itself in a different pattern each time it infects a new mass, becoming increasingly difficult to detect.

# 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 “**Mr. Zeeshan Javed**” and our **Coordinator “Dr. Saif ur Rehman”** for personal supervision, advice, valuable guidance and completion of this project. We are deeply indebted to him 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.

Anjum Shehzad

Haris Aamir

Zohaib Ali Hassan

-----

-----

-----

# Abbreviations

<b>PC</b>	Personal computer
<b>MS-DOS</b>	Microsoft disk operating system
<b>SQL</b>	Structure query language
<b>API</b>	Application programming interface
<b>LSTM</b>	Long short-term memory
<b>APT</b>	Advanced persistent threat
<b>SDLC</b>	Software development life-cycle
<b>FR</b>	Functional requirement
<b>NFR</b>	Nonfunctional requirement
<b>ENV</b>	Environment
<b>VT</b>	Virus total
<b>JSON</b>	Java script object notation
<b>CSV</b>	Comma separated values
<b>UML</b>	Unified modeling language
<b>IDE</b>	Integrated development environment
<b>ML</b>	Machine learning
<b>AI</b>	Artificial intelligence
<b>GUI</b>	Graphical user interface

# Table of Contents

<b>Chapter 1: Introduction .....</b>	<b>1</b>
1.1 Brief .....	1
1.2 Relevance to Course Modules .....	2
1.3 Project Background .....	2
1.4 Literature Review .....	4
1.5 Methodology and Software Life Cycle.....	6
<b>Chapter 2: Problem Definition .....</b>	<b>7</b>
2.1 Problem Statement.....	7
2.2 Proposed Architecture .....	7
2.3 Proposed Solution.....	7
2.4 Project Deliverables.....	7
2.5 Operating Environment .....	8
<b>Chapter 3: Requirement Analysis.....</b>	<b>9</b>
3.1 Functional Requirments.....	9
3.2 Non – Functional Requirments.....	9
3.2.1 Usability.....	10
3.2.2 Reliability .....	10
3.2.3 Integrity.....	10
3.3 Use case Model.....	11
3.3.1 Use Case Diagarm .....	12
3.3.2 Actors Discription.....	12
3.3.3 Use Case Discription .....	13
<b>Chapter 4: The Design.....</b>	<b>14</b>
4.1 UML Structural Diagrams .....	14
4.1.1 Component Diagram.....	15
4.1.2 Package Diagram .....	16
4.1.3 Deployment Diagram.....	17
4.2 UML Behavioral Diagrams .....	18
4.2.1 Activity Diagrams.....	19

4.3	UML Interaction Diagrams.....	24
4.3.1	Sequence Diagrams .....	24
<b>Chapter 5: Implementation .....</b>		<b>25</b>
5.1	Tools and Technologies .....	25
5.1.1	Python .....	25
5.1.2	Tensor Flow .....	25
5.1.3	Network and Prortocols .....	25
5.2	User Interfaces .....	26
5.3	Upload Datasets .....	27
5.3.1	Extracting Dataset.....	27
5.3.2	Uploading Files to VT .....	28
5.3.3	Extract Features .....	29
5.3.4	Displaying Features .....	30
5.3.5	Classification .....	31
5.3.6	Result Final.....	32
<b>Chapter 6: Testing and Implementation .....</b>		<b>33</b>
6.1	Module Teting .....	33
6.2	Integration Testing.....	38
<b>Chapter 7: Conclusion and Future Work .....</b>		<b>39</b>
7.1	Conclusion .....	39
7.2	Future Work.....	39
<b>References.....</b>		<b>40</b>



# List of Figures

Fig 3.3.1	Use Case Diagram .....	12
Fig 4.1.1	Component Diagram .....	15
Fig 4.1.2	Package Diagram.....	16
Fig 4.1.3	Deployment Diagram .....	17
Fig 4.2.1	Activity Diagram.....	18
Fig 4.2.1.1	Data Extraction Diagram.....	19
Fig 4.2.1.2	Report Generation Diagram .....	20
Fig 4.2.1.3	Feature Extraction Diagram .....	21
Fig 4.2.1.4	Train Model Diagram.....	22
Fig 4.2.1.5	Classification Diagram .....	23
Fig 4.3	Sequence Diagram .....	24

# List of Tables

Table 3.1 Functional Requirements .....	8
Table 3.2 Non-Functional Requirements.....	9
Table 3.3.3 Use Cases.....	10
Table 3.3.3.1 Submit Malware samples.....	11
Table 3.3.3.2 Virus Total Environment .....	12
Table 3.3.3.3 Feature Extraction.....	13
Table 3.3.3.4 Training Model .....	14
Table 3.3.3.5 Classification .....	15
Table 6.1.1 Check Virus Total Configuration .....	33
Table 6.1.2 Upload Malware Samples.....	34
Table 6.1.3 Collect JSON Reports.....	35
Table 6.1.4 Feature Extraction.....	36
Table 6.1.5 Training Mode .....	37
Table 6.1.6 Classification .....	38

# Chapter 1: Introduction

## 1.1. Malware:

"Malware" is an abbreviation for "malicious software", it is used as a single term to refer to Viruses, Trojans, Worms, etc. These programs have a variety of features, such as stealing, encrypting or deleting sensitive data, modifying or hijacking basic computer functions, and monitoring computer activity. Show user permission.

## 1.2. Computer virus:

It is generally a program that is installed outside the user's will and can cause damage to both the operating system and the hardware (physical) elements of a computer. Effects generated by the virus:

- Destruction of files.
- Changing the file size.
- Delete all information on the disc, including formatting it.
- Destruction of the file allocation table, which makes it impossible to read the information on the disk.

## 1.3. Worms:

Computer worms are programs with destructive effects that use communication between computers to spread. Worms have common features with viruses, i.e., Worms are able to multiply like viruses, but not locally, but on other computers. It uses computer networks to spread to other systems.

Types of computer worms:

- E-Mail worms.
- Instant messaging worms.
- Internet worms.

## **1.4. Ransomware:**

Ransomware is a type of malware that blocks the victim's access to the computer and demands payment of a reward. The reward and the official reason why the victim should pay depends on the type of virus. Some versions of ransomware claim that the payment should be made to avoid punishment by a government authority (usually the local agency), others inform that this is the only way to decrypt encrypted data.

### **What spyware can be used for?**

- To steal sensitive information. Such programs are interested in personal information, such as credentials, passwords, bank details, and other similar information. In addition, they can monitor the user's online activity, track their web browsing habits, and send all this data to a remote server.
- Show unwanted creatives. Spyware can display a large number of annoying pop-up ads. Such activity is more associated with adware parasites.
- Redirecting users to questionable or malicious websites against their will. In addition, some types of spyware threats are able to change web browser settings and change the search engine and home page.
- Create numerous links in the search results of the victim and redirect him / her to the desired places (third party spyware sites, websites and other associated fields).

## **1.5. Relevance to Course Modules:**

Our project is related to various courses we have studied in our degree which are mentioned below:

- Object oriented Programming.
- Artificial Intelligence.
- Software Engineering-I.

## **1.6. Project Background:**

The first versions of Malware were primitive, they infested various machines through floppy disks. With the evolution of Networking and the maturation of the Internet, malware authors have adapted their malicious codes to take full advantage of this new communication environment. Below is a brief overview of the evolution of malware over time.

### **1.6.1. The Years 1971-1999:**

- 1971-Creeper: An experiment designed to test how a program can move between computers.
- 1974-Wabbit: A program that multiplies itself at an accelerated pace, until the speed of the system slows down, the performance is measured, the system is reduced and eventually collapses.
- 1982-Elk Cloner: Written by a 15-year-old child, Elk Cloner is one of the first viruses, and widespread, to multiply itself and display a short "poem" to the infected person: "It will get on all your disks; It will infiltrate your chips; Yes, it's a Cloner."
- 1986-Brain Boot Sector Virus: Considered the first virus to infect MS-DOS computers.
- 1986 — PC-Write Trojan: Malware disguised as one of the oldest Trojans as a popular program called "PC-Writer." Once on a system, it deletes all files of a user.
- 1988 — Morris Worm: Infected a substantial percentage of computers connected to ARPANET, the predecessor of the Internet, which brought the network to its knees in 24 hours. This Worm marked a new beginning for malicious software.
- 1991 — Michelangelo Virus: The virus was designed to erase information from hard drives on March 6, the birthday of the famous Renaissance artist.
- 1999 - Melissa Virus: used Outlook addresses from infected machines and was sent to 50 people at once.

### **1.6.2. The Years 2000-2010:**

- 2000 – ILOVEYOU Worm: the worm infected about 50 million computers. The damage caused major corporations and government agencies, including portions of the Pentagon and the British Parliament, to shut down their e-mail servers. Worms have spread globally and cost more than \$ 5.5 billion in damage.
- 2003 – SQL Slammer Worm: One of the fastest spreading worms of all time, SQL Slammer infected nearly 75,000 computers in ten minutes. The worm has had a major effect worldwide, slowing down worldwide Internet traffic by denial of service.
- 2004 – Cabir Virus: Although this virus has caused some damage, it is noteworthy because it is widely recognized as the first cell phone virus.
- 2005 – Koobface Virus: One of the first cases of malware to infect PCs and then spread to social networking sites. If rearranged, the letters in "Koobface" are old, and you get "Facebook". The virus has also attacked other social networks such as MySpace and Twitter.
- 2008 – Conficker Worm: A combination of the words “configure” and “ficker”, this sophisticated worm has caused some of the 11 worst damage observed since Slammer appeared in 2003.

### **1.6.3. The Year 2010 - Present:**

- 2010 – Stuxnet Worm: Shortly after its release, security analysts openly speculated that the malware was designed to explicitly attack Iran's nuclear program and include the ability to affects hardware and software. The incredibly sophisticated worm is considered to be the work of a whole team of developers, making it one of the most intensive malware resources created to date.
- 2011 — Zeus Trojan: Often detected for the first time in 2007, the author of the Trojan Zeus released the code to the public in 2011, giving a new life to malware. Sometimes called the Zbot, this Trojan has become one of the most successful pieces of botnet software in the world, impacting millions of machines.

- 2013 – Cryptolocker: had a significant impact globally and helped fuel the ransomware era.
- 2014 – Backoff: Malware designed to compromise Point-of-Sale (POS) systems to steal credit card data.
- 2016 – Cerberus: One of the most prolific crypto-malware threats. At one point, Microsoft found more company PCs infected with Cerberus than any other family of ransomware.
- 2017 – WannaCry Ransomware: Exploiting a vulnerability first discovered by the National Security Agent, WannaCry Ransomware has knelt down a number. systems in Russia, China, the United Kingdom and the United States, blocking access to data and demanding redemption or loss of everything. The virus has affected at least 150 countries, including hospitals, banks, telecommunications companies, warehouses and many other industries.

## **1.7. Literature Review:**

In this chapter we will go through the previous work which has been done in this field. We will discuss some of the projects which have been made and what was the purpose behind the creation and what benefits did they give to the field.

The list of the projects which we will be discussing is given below:

- Deep learning based Sequential model for malware analysis using Windows Exe API Calls.
- Optimal feature configuration for dynamic malware detection.
- Toward Identifying APT Malware through API System Calls.

### **1.7.1. Deep learning Based Sequential Model for Malware Analysis using Windows exe API calls:**

This study is focused on metamorphic malware, which is the most advanced member of the malware family. It is quite impossible for anti-virus applications using traditional signature-based methods to detect metamorphic malware, which makes it difficult to classify this type of malware accordingly. Recent research literature about malware detection and classification discusses this issue related to malware behavior. The main goal of this project is to develop a classification method according to malware types by taking into consideration the behavior of malware. This search was started by developing a new dataset containing API calls made on the windows operating system, which represents the behavior of malicious software. The types of malicious malware included in the dataset are Adware, Backdoor, Downloaded, Dropper, spyware, Trojan, Virus, and Worm. The classification method used in this study is LSTM (Long Short-Term Memory), which is a widely used classification method in sequential data. The results obtained by the classifier demonstrate accuracy up to 95% with 0.83  $F_1$ -score, which is quite satisfactory. Recent research literature about malware detection and classification discusses this issue related to malware behavior. The main goal of this project is to develop a classification method according to malware types by taking into consideration the behavior of malware. This search was started by developing a new dataset containing API calls made on the windows operating system, which represents the behavior of malicious software. We also run our experiments with binary and multi-class malware datasets to show the classification performance of the LSTM model.

### **1.7.2. Optimal Feature Configuration for Dynamic Malware Detection:**

Applying machine learning techniques to malware detection is a common approach to try to overcome the limitations of signature-based methods. However, it is difficult to engineer a set of features that characterizes the samples properly, especially when various file types may be a vector of infection. In this work, researchers configured several feature sets for dynamic malware detection extracted



from API calls, including an alternative scheme grouping calls in categories, network activity, signatures from the Cuckoo sandbox report, and some interactions with the file system and registry. They tested the combinations of these feature sets to ascertain whether they are good enough to distinguish between benign and malicious samples from a dataset containing several file types, obtained from public sources and applied statistical inference to measure the differences in the performance between the feature sets, and the hyper parameter optimization algorithms applied to construct the models. They also unbalance the datasets to evaluate the model performance on more realistic scenarios in which not many malware samples are available. Although all studied feature configurations provide accuracies greater than 0.98, and several of them a Matthews correlation coefficient greater than 0.95 in the unbalanced datasets, statistically meaningful differences appear, so they analyzed the results to determine which is the optimal set of features. They obtain a model that achieves an accuracy of 0.9937 in the balanced dataset and a Matthews correlation coefficient of 0.964 in the unbalanced dataset with 5% of malware.

### **1.7.3. Toward Identifying APT Malware through API System Call:**

Self-developed malware was usually used by advanced persistent threat (APT) attackers to launch APT attacks. Therefore, we can enhance the understanding and cognition of APT attacks by comprehending the behavior of APT malware. Unfortunately, the current research cannot effectively explain the relationship between the recognition, detection, and defense of APT. The model of similar studies also lacks an explanation about it. To defend against APT attacks and inquire about the similarity of different APT attacks, this study proposes an APT malware classification method based on a combination of multiple deep learning algorithms and transfer learning by collecting malware used in several famous APT groups in public. By extracting the application programming interface (API) system calls, with the vector representation of features by combining dynamic LSTM and attention algorithm, we can obtain API at different APT family's classification contributions trained dynamic. Thus, we used transfer learning to

perform multiple classifications of the APT family. This study aims to reduce the burden of network security staff from reviewing a large number of suspicious files when defending against APT attacks. Additionally, it can effectively intercept them in the initial invasion stage of APT to perform targeted defense against specific APT attacks by combining threat intelligence in public. The experimental result shows that the proposed method can achieve 99.2% in distinguishing common malware from APT malware and assign APT malware to different APT families with an accuracy of 95.5%.

### **1.8. Methodology and Software Life-cycle 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

## 2.1. Problem Statement:

Applying machine learning techniques to malware detection is a common approach to try to overcome the limitations of signature-based methods. Signature based detection uses the technique of pattern recognition in search of predefined signatures stored in a database. Signature based detection technique is commonly used up by Anti-viruses as their basic block. This technique does not stand on its hundred percent as it is unable to detect malware whose signature is not present in its database record. These methods are unable to detect unknown malware variants and also requires high amount of manpower, time, and money to extract unique signatures. These are the main disadvantages of these methods.

## 2.2. Purposed Architecture:

- 1) Upload dataset on Virustotal website.
- 2) Extract Features from malware reports.
- 3) Test and train model
- 4) Perform Classification

## 2.3 Proposed Solution:

To solve the problem, we configure several feature sets for dynamic malware detection extracted from API calls, including an alternative scheme grouping calls in categories, network activity, signatures from the Cuckoo sandbox report, and some interactions with the file system and registry. API call sequences is one of the most attractive ways that reflects the behavior of a piece of code like malware. We test combinations of these feature sets to ascertain whether they are good enough to distinguish between benign and malicious samples from a dataset containing several file types, obtained from public sources.

## **2.4 Project Deliverable:**

- Documentation
- Features Extraction
- Desktop Application
- Classification System

## **2.5 Operating Environment:**

- Virustotal
- Windows 10
- Ubuntu

# Chapter 3: Requirement Analysis

**Table 3.1: Functional Requirement**

<b>Functional Requirement No.</b>	<b>Functional Requirement Description</b>
FR.1	User Collect Malware Samples.
FR.2	Submit Malware Samples in sandbox environment.
FR.3	Extract features from these reports and prepare a CSV file.
FR.4	After training the model classify the malware in families.

**Table 3.2: Non Functional Requirement**

<b>Non-Functional Requirement No.</b>	<b>Non-Functional Requirement Description</b>
NFR.1	System will provide interface by using Tkinter.
NFR.2	System will take sandbox generated malware reports.
NFR.3	It will extract features from reports and prepare CSV File.

### **3.3. 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.

### **3.4. Reliability:**

This system should provide appropriate answers to the user. This system should be able to interact efficiently with the user.

### **3.5. Integrity:**

This desktop application will require specific python version to run. It also requires an active internet connection to work and to exchange queries to provide information to the user.

### **3.6. Use Case Model:**

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.

### 3.7 Use Case Diagram:



**Figure 3.7: Use Case Diagram**

### 3.8. Actor Description:

In our project we only have one actor who will be using the system directly. The user will interact with the system and get the work done himself. The user will provide the malicious files to the system directly and he will get the classification done from the system right away.



### 3.9. Use Case Description:

**Table 3.9.1: Submit Malware Data Samples.**

Use case ID	U_id:1
Use case Name	Submit Malware Data Samples.
Actor Name	Direct User.
Description	The data-set is selected and uploaded on the system.
Trigger	To accomplish this task we Click on button (Upload Data Set).
Precondition	NO info about samples.
Post Condition	Data set must be uploaded on the system.
Normal Flow	Choose the malware samples from system.
Alternative Flow	If successfully 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.Uploading Error

**Table 3.9.2: Virus total Environment.**

Use case ID:	U_id:2
Use case Name	Virus total Environment.
Actor Name	Direct User.
Description:	After submitting the malware to Virus Total. Then Virus Total will run those samples in isolated virtual environment and generate a report in JSON format.
Trigger	For this process user upload the files on Virus Total and it will run samples on isolated environment and generate a report at the end.
Precondition	Isolated environment must be configured with Virus Total and in restore condition.
Post Condition	Generate report in CSV form which is downloaded by the user.

Normal Flow:	1. Run malware samples in isolated environment. 2. Generate a report based on executed malware samples.
Alternative Flow:	If downloaded successfully then move to the next step else download again.
Exception:	During the performing action some error can be take place: 1. Virus Total is not configured with isolated environment properly.
Includes:	Report Generation based on executed malware sample.

**Table 3.9.3: Feature Extraction.**

<b>Use case ID:</b>	<b>U_id:3</b>
Use case Name	Feature Extraction.
Actor Name	Direct User.
Description:	After the report is generated based on executed malware sample. Next step is to extract features from report in JSON form.
Trigger:	To perform this task a complete report should be collected from Virus Total.
Precondition	No feature is extracted.
Post condition	After extracting features save in CSV file.
Normal Flow	1. Get report from Virus Total. 2. Extract features from reports. 3. Make a CSV file of features which are extracted from report.
Alternation	Successfully move next else again try to extract features from report.
Exception:	During the action some error can take place 1. Index number of desired feature is not defined or correct. 2. Desired feature may not exist. 3.Fail to Extract features
Includes:	Null

**Table 3.9.4: Training Model.**

Use case ID	U_id:4
Use case Name	Training Model.
Actor Name	Direct User.
Description	The algorithms are designed in the system to train the model.
Trigger	To accomplish this task, we have to run those algorithms.
Precondition	Extract the features from the JSON report and according to the features model will be train.
Post Condition	Algorithm must be working properly.
Normal Flow	The files must be passed from the trained model.
Alternative Flow	If the model is trained successfully otherwise check the algorithms again.
Expectation	During the implementing algorithm the trained model must give the appropriate results otherwise in case of any error algorithm must be rechecked.
Includes:	Null

**Table 3.9.5: Classification.**

Use case ID:	U_id:5
Use case Name	Classification.
Actor Name	Direct user.
Description	For Classification of Malware first extract features then prepare CSV based on these extracted features. After that perform classification using deep learning model.
Trigger	To perform this step press classification button.
Precondition	Non-Classified data-set
Post Condition	Classified.
Normal Flow	Get CSV file

	Classify the files that it is malicious or non-malicious.
Alternative	If success then move to the final steps else go back for classification
Exception	Data-set is not prepared properly.
Includes	Null.

## Chapter 4: Design and Architecture

### 4.1 UML Structural Diagrams:

UML Structural Diagrams depict the elements of a system that are independent of time and that convey the concepts of a system and how they relate to each other.

#### 4.1.1 Component Diagram:

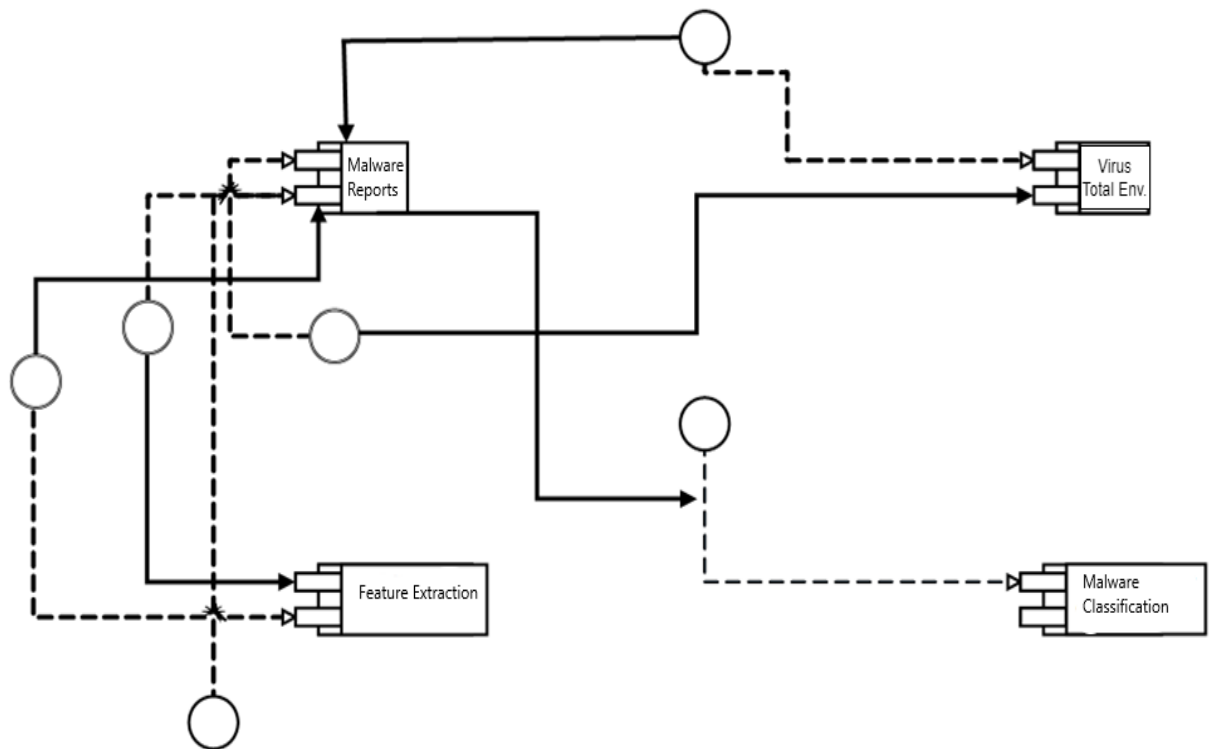
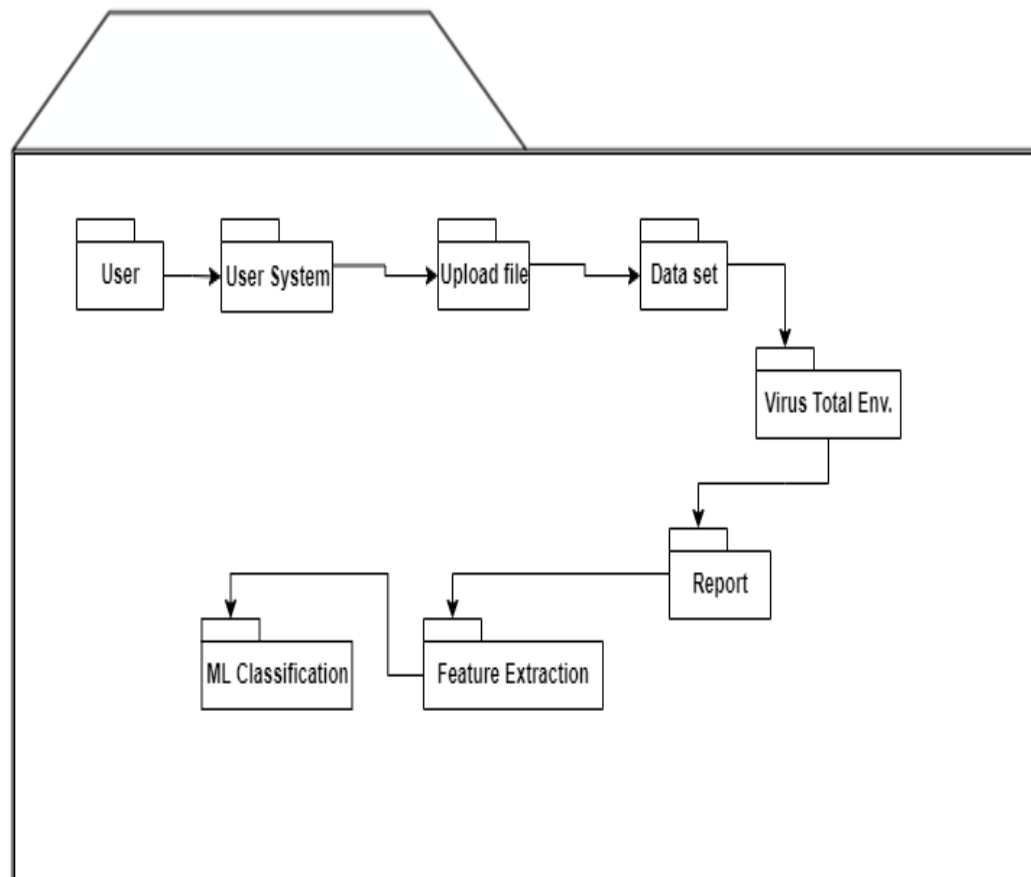


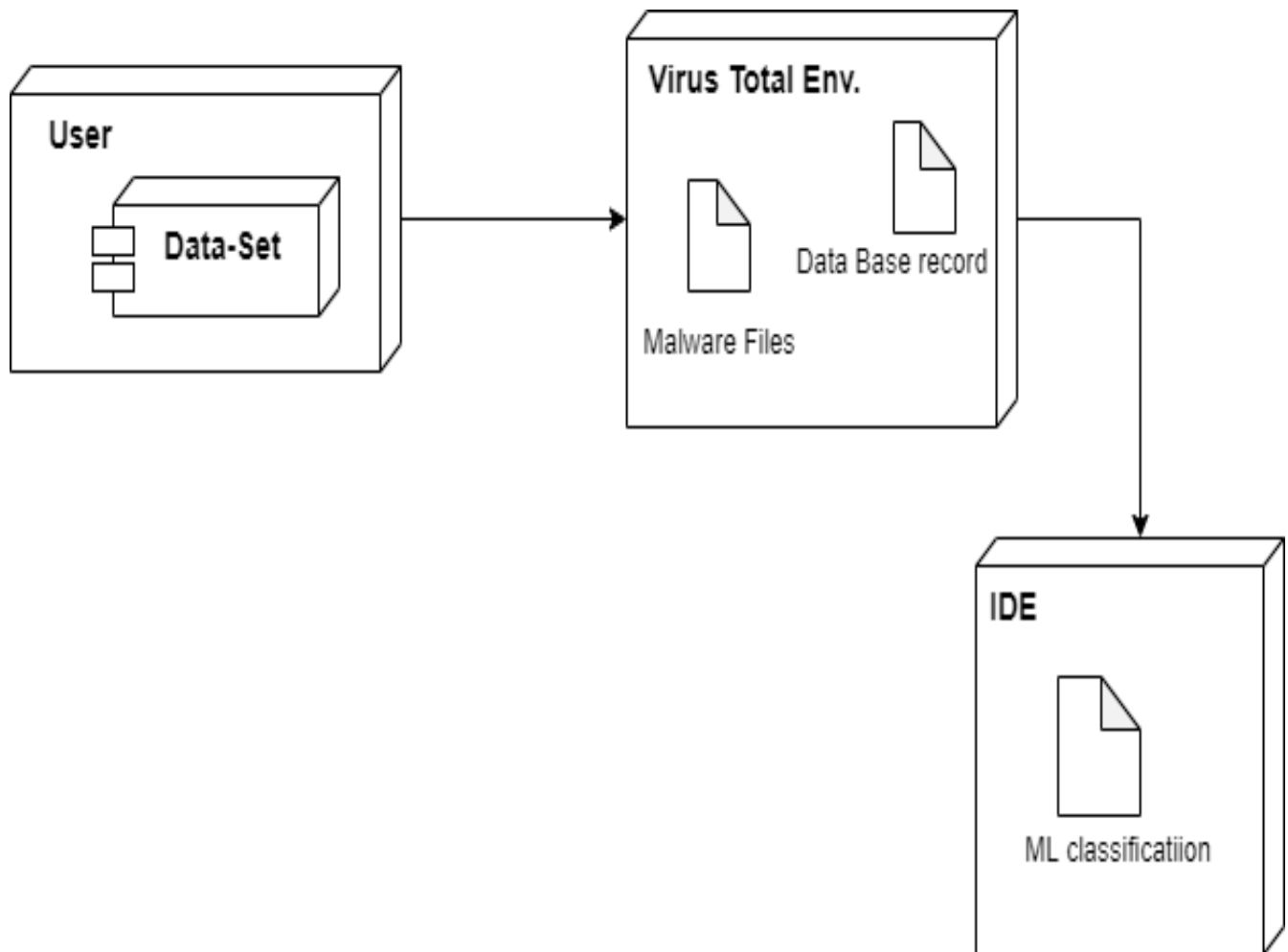
Figure 4.1.1: Component Diagram

#### 4.1.2 Package Diagram:



**Figure 4.1.2: Package Diagram**

#### 4.1.3 Deployment Diagram:



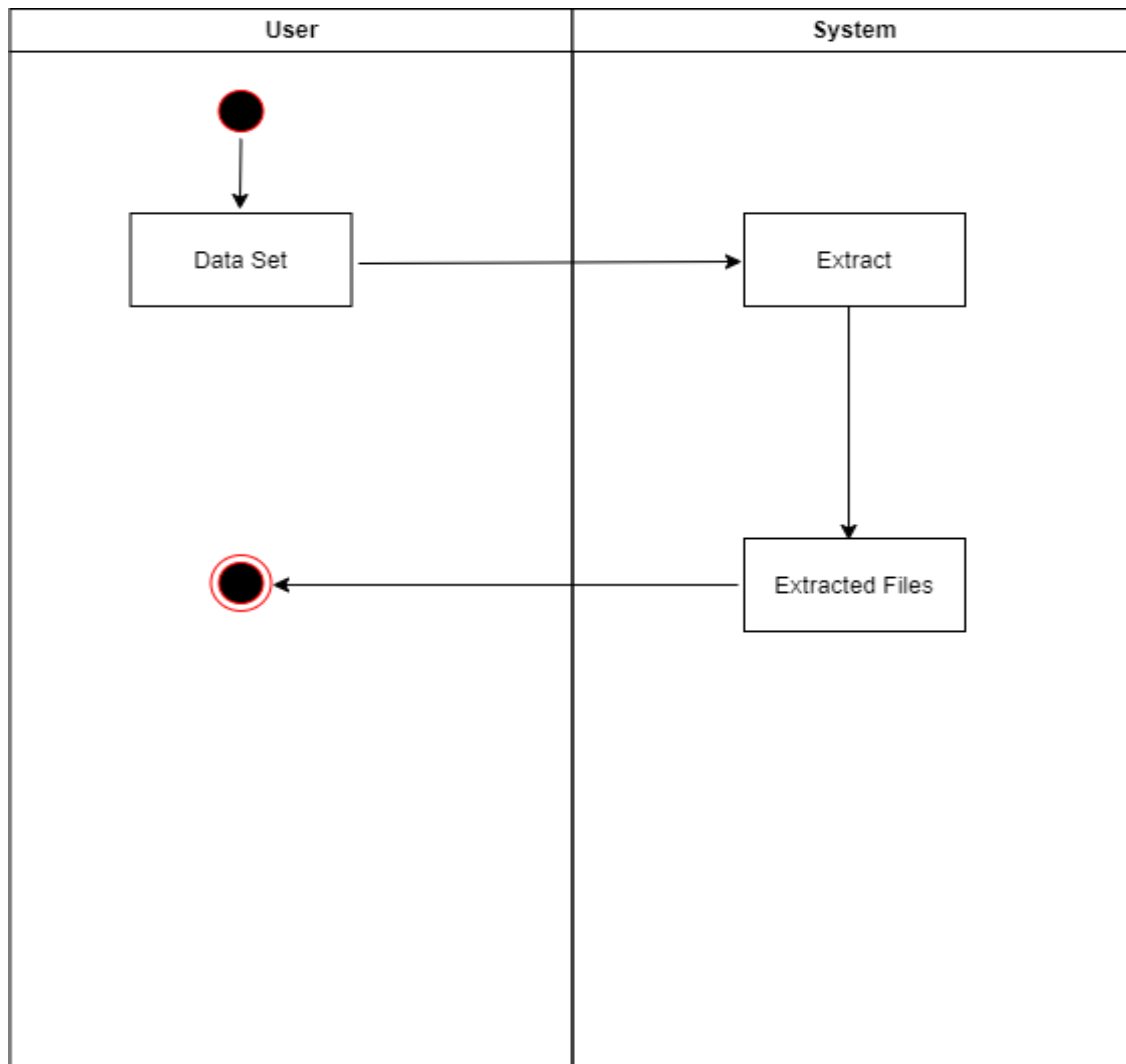
**Figure 4.1.3: Deployment Diagram**

## 4.2. UML Behavioral Diagrams:

### 4.2.1. Activity Diagram:

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system. Activity diagram is basically a flowchart to represent the flow from one activity to another activity.

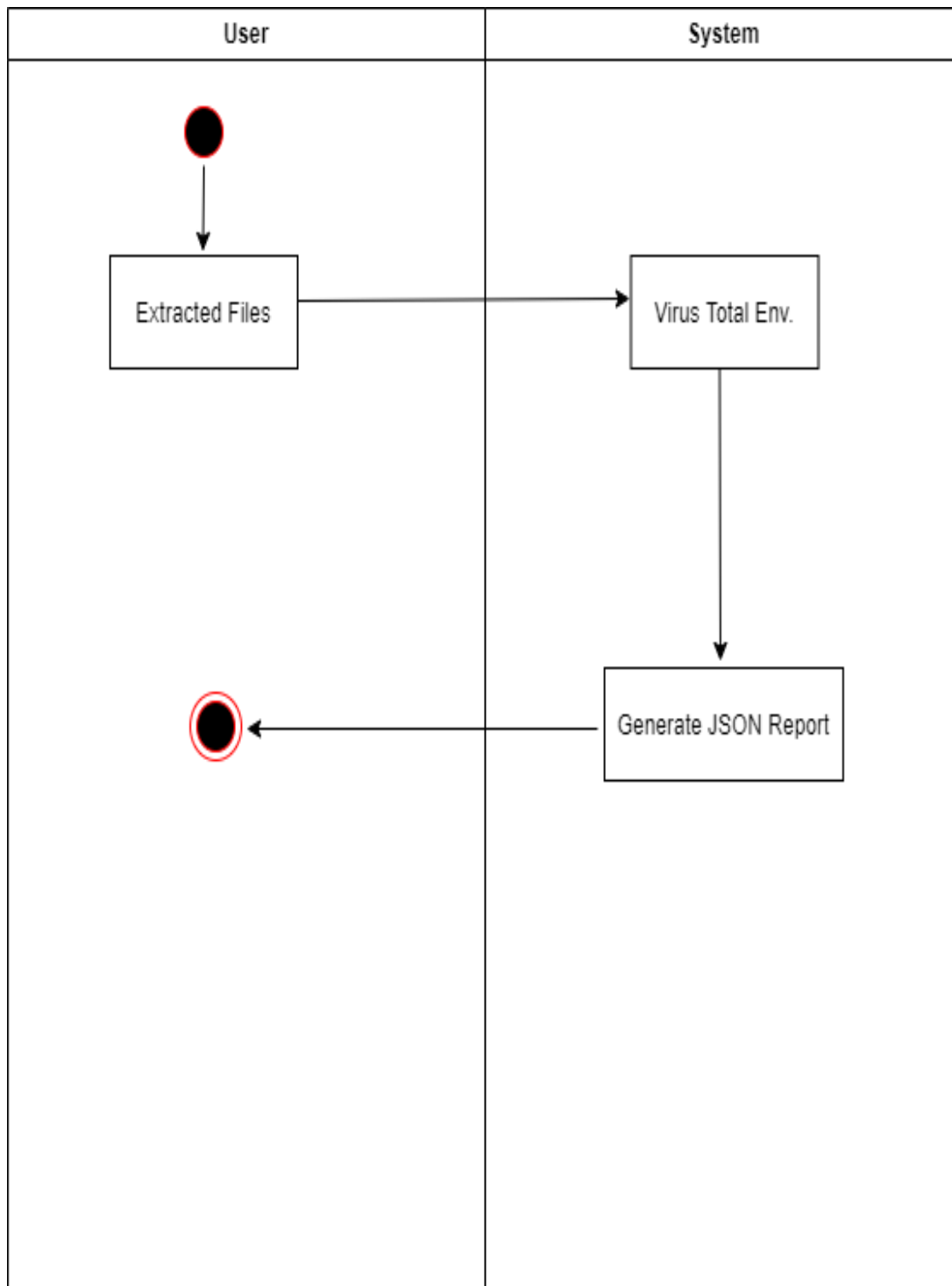
#### 4.2.1.1. Data Extraction:



**Figure 4.2.1.1: Data Extraction**

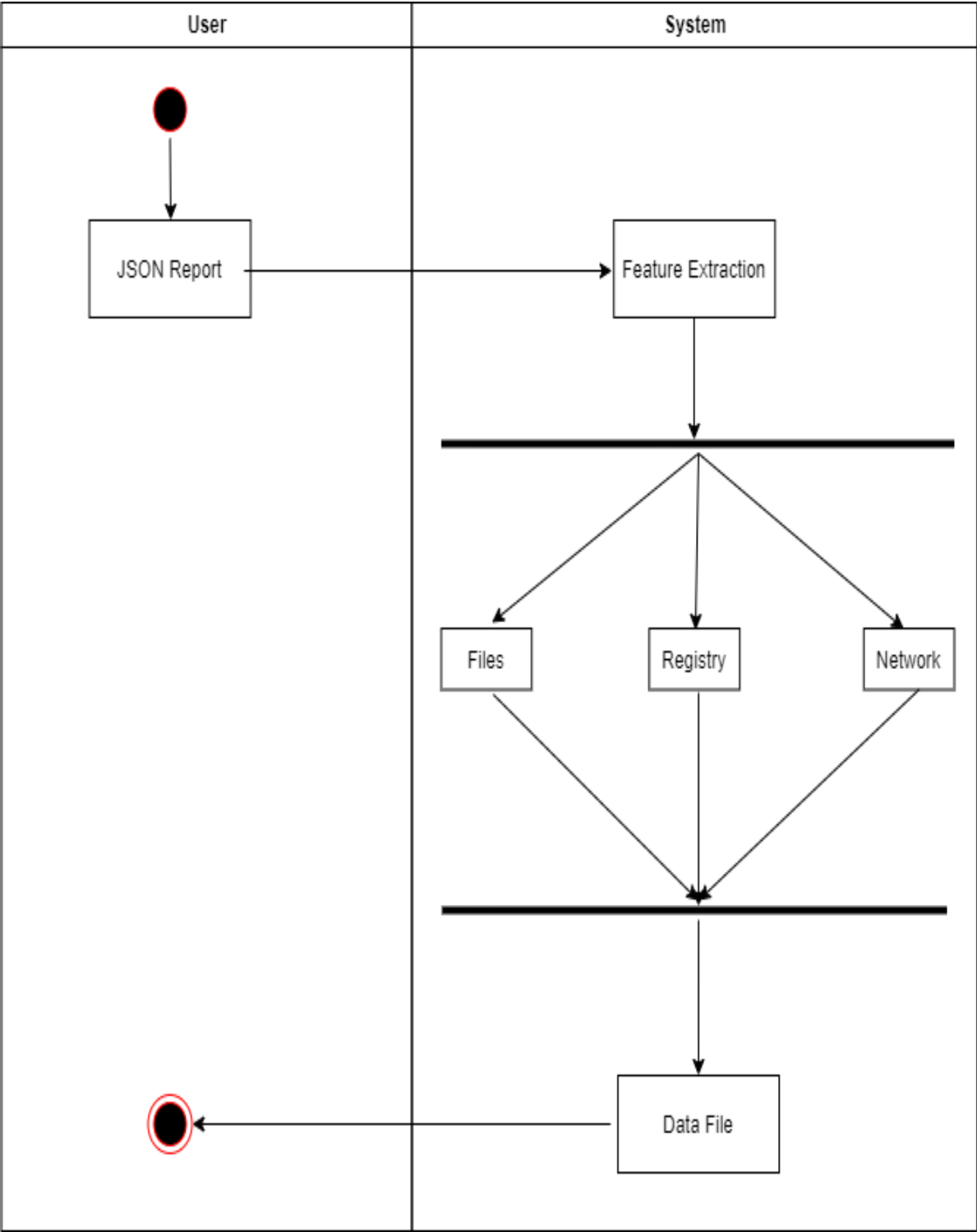


#### 4.2.1.2. Report Generation:



**Figure 4.2.1.2: Report Generation**

**4.2.1.3 Feature Extraction:**



**Figure 4.2.1.3: Feature Extraction**

#### 4.2.1.4 Train Model:

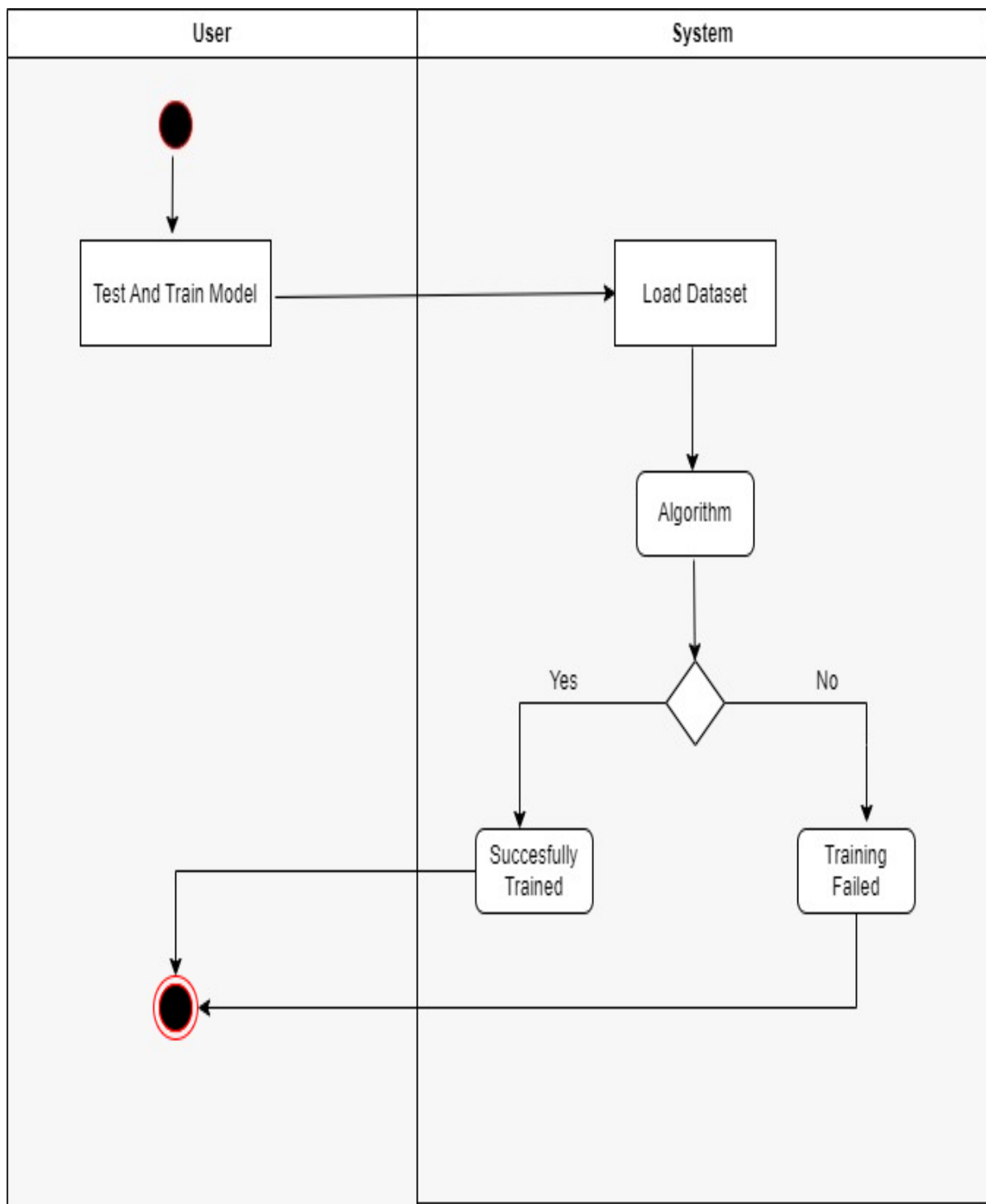
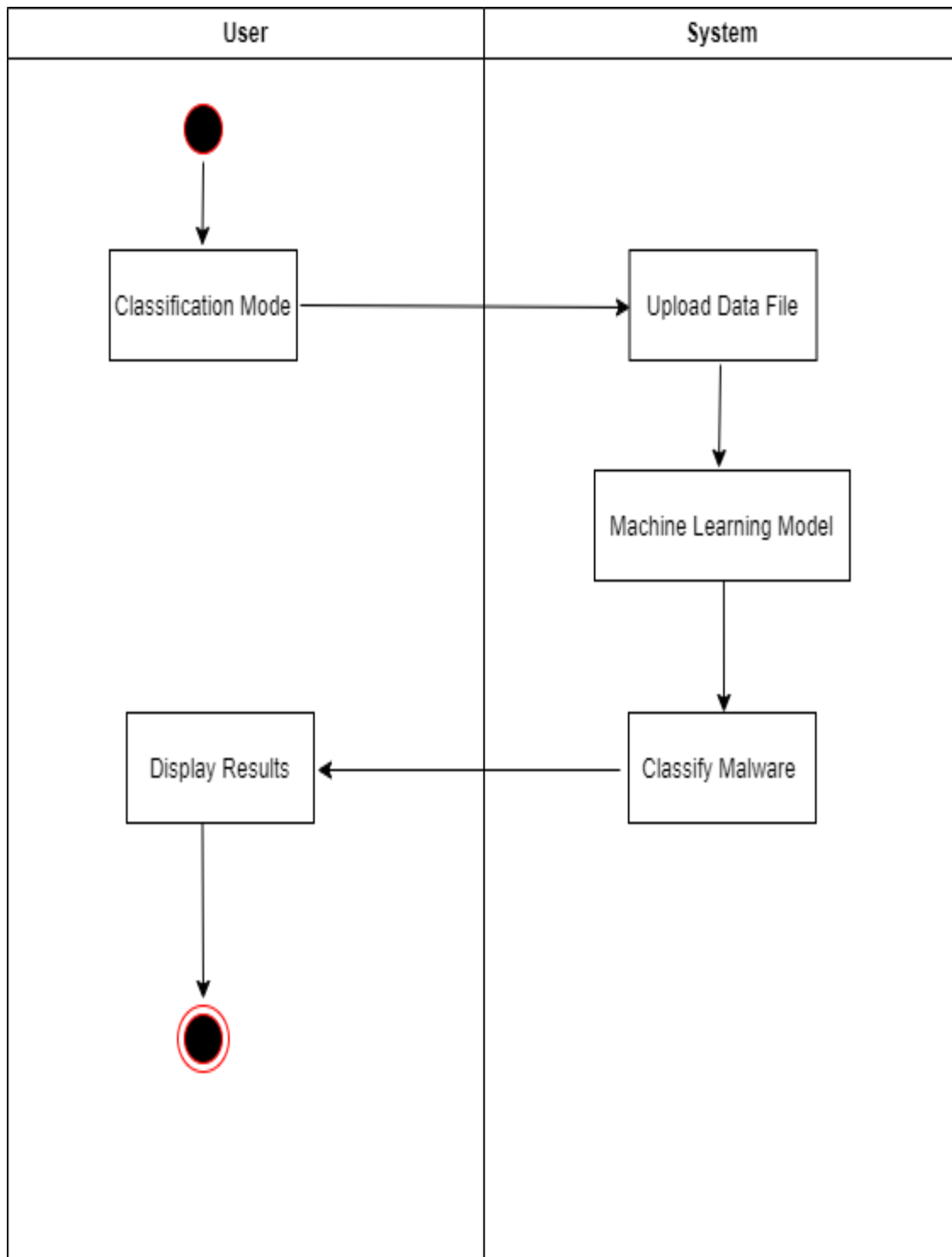


Figure 4.2.1.4: Train Model

#### 4.2.1.5 Classification Model:

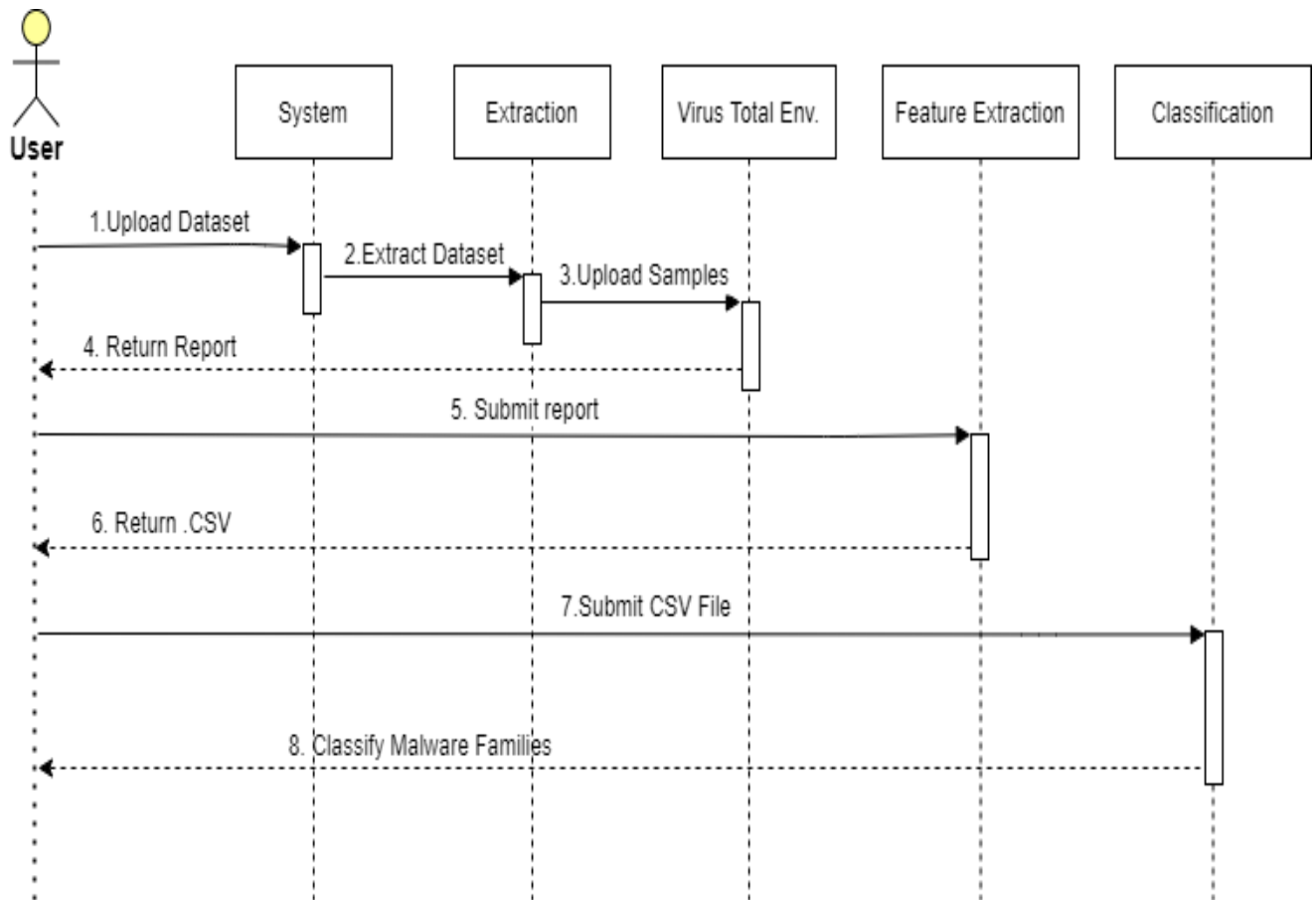


**Figure 4.2.1.5: Classification Model**

## 4.3 UML Interaction Diagrams:

### 4.3.1 Sequence Diagram:

A sequence diagram is a type of interaction diagram because it describes how and in what order a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process.



**Figure 4.3.1: Sequence Diagram**

# Chapter 5: Implementation

## 5.1. Tools and Technologies:

Tools and technologies which are being used in the project creation according to the requirement of the system.

## 5.2. Python:

Python is a general-purpose programming language. We used Python because it is the best fit for machine learning, deep learning and AI-based projects. We used it because of consistency and access to great libraries and frameworks for machine learning.

## 5.3. Tkinter:

Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows.

## 5.4. Tensorflow:

It is an open source artificial intelligence library, using data flow graphs to build models. We used TensorFlow mainly for: Classification, Perception, Understanding, Discovering, Prediction and Creation.

## 5.5. Network and Protocol Choice:

- **Design:** The designing of this malware classifier and the diagram have been done through MS-Visio.
- **Tkinter:** Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit and is Python's de facto standard GUI.
- **Tensorflow:** TensorFlow is a free and open-source software library for machine learning.

## 5.6. User Interface:

### 5.6.1 Uploading Dataset:

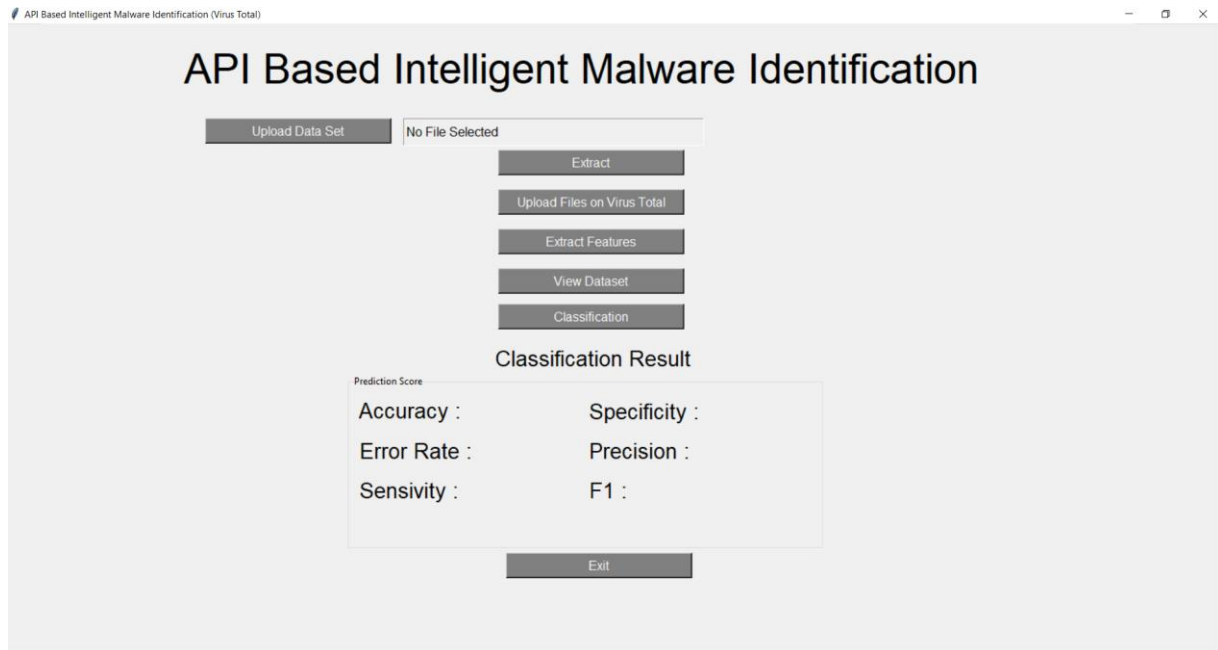


Figure 5.6.1: Uploading Dataset

### 5.6.2 Extracting Dataset:

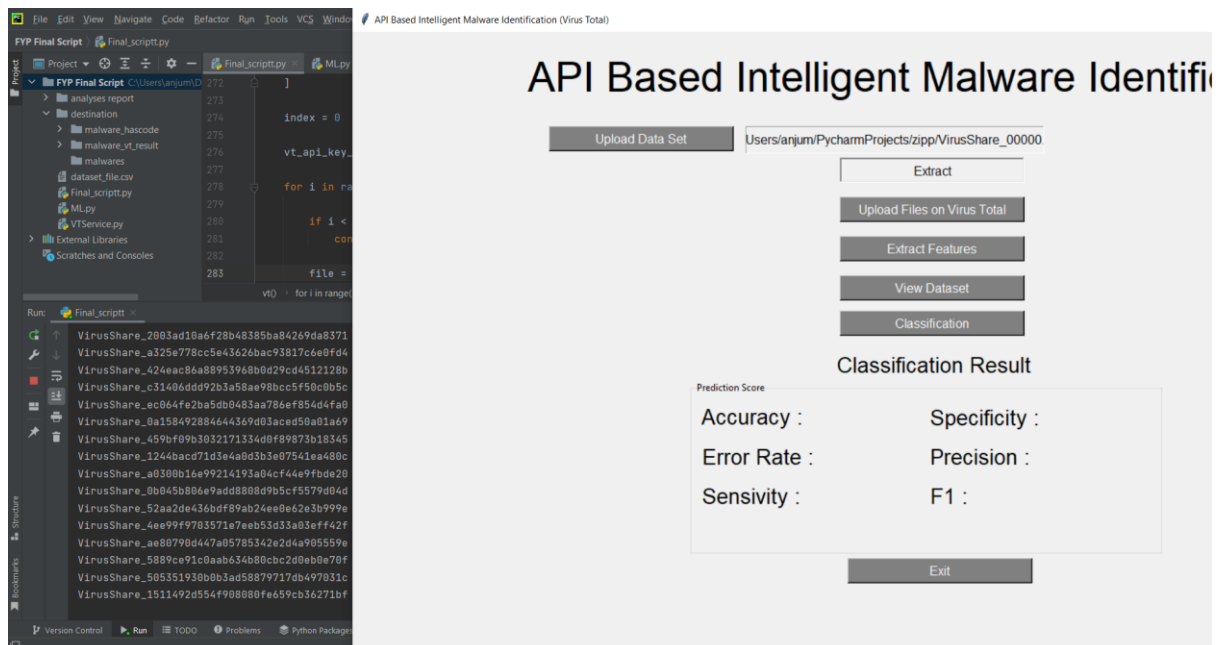


Figure 5.6.2: Extracting Dataset

### 5.6.3 Uploading Files on Virus Total Website:

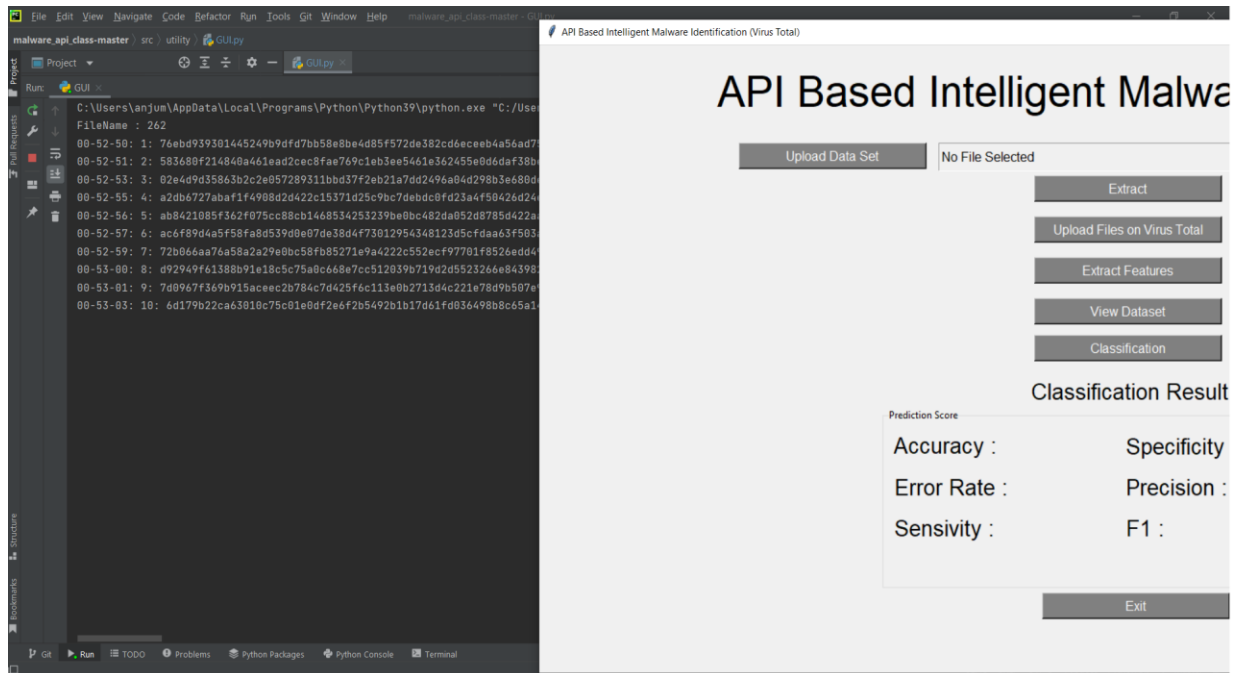


Figure 5.6.3: Uploading Files on Virus Total Website

### 5.6.4 Extracting the features:

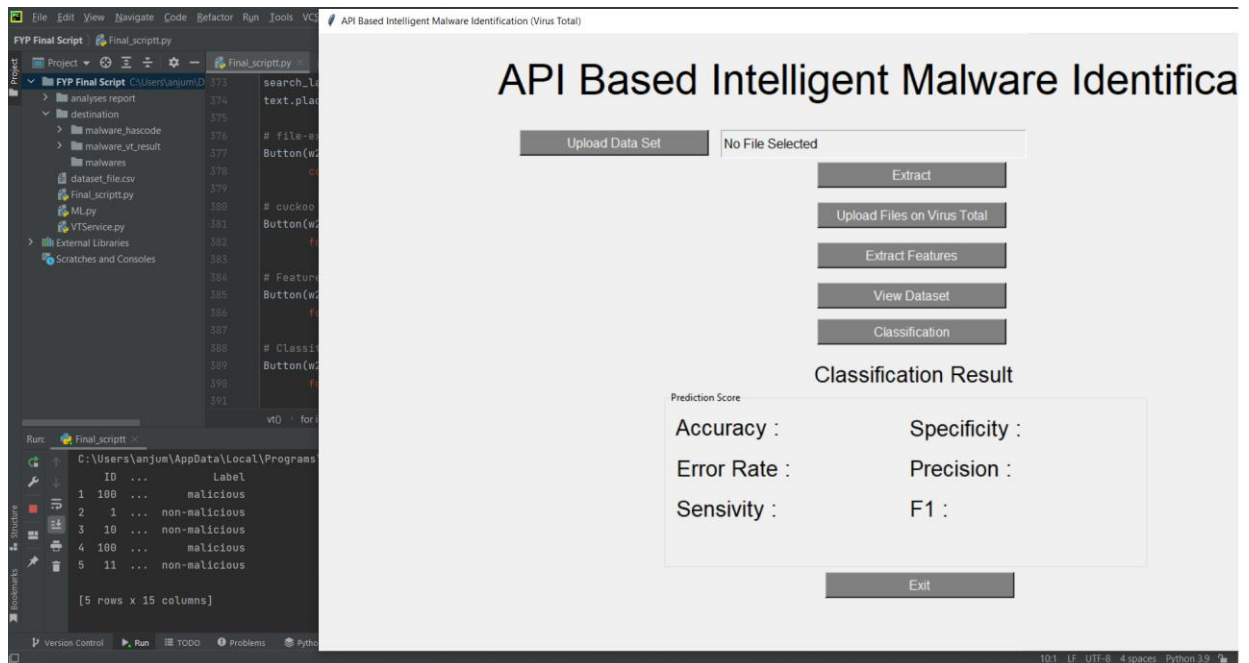
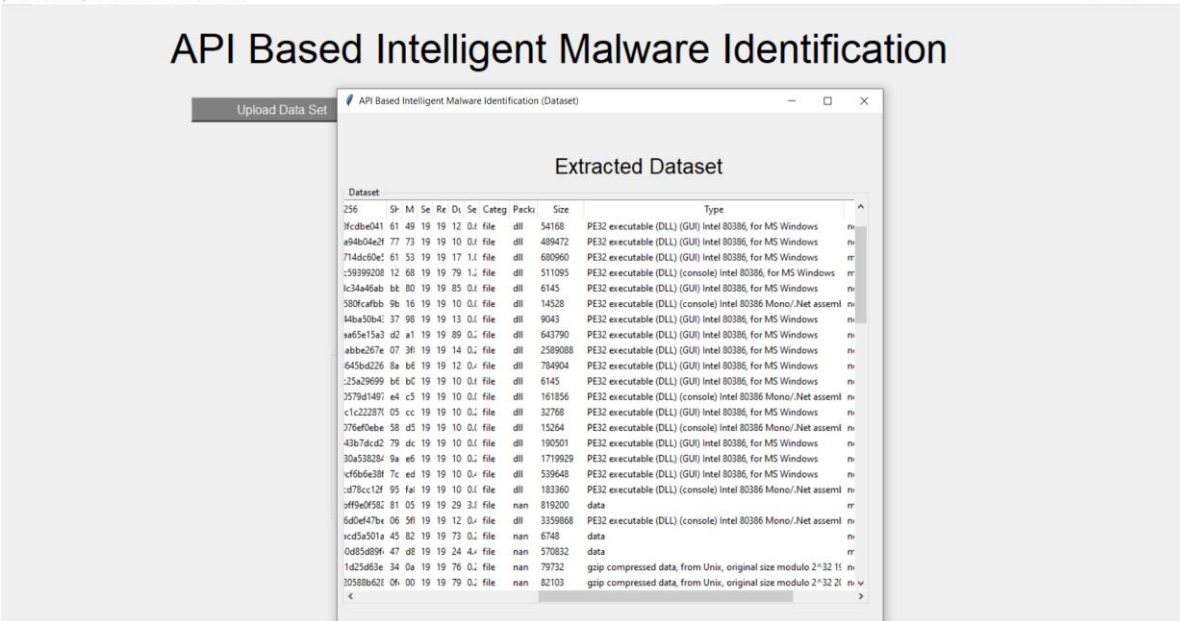


Figure 5.6.4: Extracting the features



## 5.6.5 Displaying the Dataset of Extracted features:

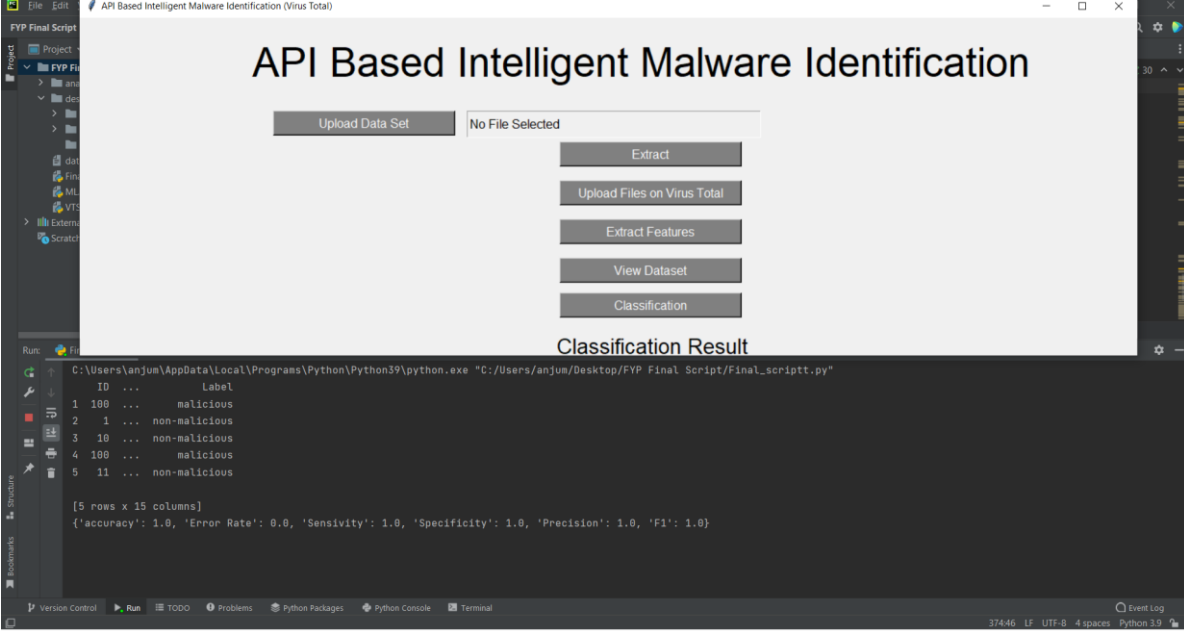


The screenshot shows a window titled "API Based Intelligent Malware Identification (Dataset)". Inside, there is a table titled "Extracted Dataset". The table has columns: Dataset, Sh, M, Se, Re, Di, Se, Categ, Pack, Size, and Type. The data rows show various file hashes, sizes, and types, such as "PE32 executable (DLL) (GUI) Intel 80386, for MS Windows".

Dataset	Sh	M	Se	Re	Di	Se	Categ	Pack	Size	Type
256										
Jfcdbe041	61	49	19	19	12	0.1	file	dll	54168	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
e94b04e2f	77	73	19	19	10	0.1	file	dll	489472	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
714dc0e1	61	53	19	19	17	1.1	file	dll	680960	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
-59399208	12	68	19	19	79	1.1	file	dll	511095	PE32 executable (DLL) (console) Intel 80386, for MS Windows
lc34a46ab	bb	80	19	19	85	0.1	file	dll	6145	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
580fcafb	9b	16	19	19	10	0.1	file	dll	14328	PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly
44ba50b4	37	98	19	19	13	0.1	file	dll	9043	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
xa65e15a3	d2	a1	19	19	89	0.1	file	dll	643790	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
abbe267e	07	3f	19	19	14	0.1	file	dll	2589088	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
645bd226	8a	bf	19	19	12	0.1	file	dll	784904	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
-25a29699	b6	bc	19	19	10	0.1	file	dll	6145	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
7579d1497	e4	c5	19	19	10	0.1	file	dll	161856	PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly
c1c22287f	05	cc	19	19	10	0.1	file	dll	32768	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
776ef0ebe	58	dc	19	19	10	0.1	file	dll	15264	PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly
43b7dc42	79	dc	19	19	10	0.1	file	dll	190501	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
30a538284	9a	e5	19	19	10	0.1	file	dll	1719929	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
c46b6e38f	7c	ed	19	19	10	0.1	file	dll	539648	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
-d78cc12f	95	fa	19	19	10	0.1	file	dll	183360	PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly
pf9e0f582	81	05	19	19	29	3.1	file	nan	819200	data
6d0ef47be	06	5f	19	19	12	0.1	file	dll	3359868	PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly
icd5a501a	45	82	19	19	73	0.1	file	nan	6748	data
0d85d89f	47	d8	19	19	24	4.1	file	nan	570832	data
1d25d63e	34	0a	19	19	76	0.1	file	nan	79732	gzip compressed data, from Unix, original size modulo 2^32 11
20588b62	0f	00	19	19	79	0.1	file	nan	82103	gzip compressed data, from Unix, original size modulo 2^32 21

Figure 5.6.5: Displaying the Dataset of Extracted features

## 5.6.6 Classification Result:



The screenshot shows the "API Based Intelligent Malware Identification" window with the "Classification Result" button clicked. The result is displayed in the terminal window below the main interface.

```
C:\Users\anjum\AppData\Local\Programs\Python\Python39\python.exe "C:/Users/anjum/Desktop/FYP Final Script/Final_scriptt.py"
ID ... Label
1 100 ... malicious
2 1 ... non-malicious
3 10 ... non-malicious
4 100 ... malicious
5 11 ... non-malicious

[5 rows x 15 columns]
{'accuracy': 1.0, 'Error Rate': 0.0, 'Sensitivity': 1.0, 'Specificity': 1.0, 'Precision': 1.0, 'F1': 1.0}
```

Figure 5.6.6: Classification Result

# Chapter 6: Testing and Evaluation

## 6.1. Module/Unit testing:

Here we will test the first module of the application in which GUI and its configuration is tested.

**Table 6.1: Check Virus Total Configuration.**

Project Name: API Based Intelligent Malware Detection					
Test Case: 1					
Test Case ID			T_id:01		
Test priority (low, medium, high)			High		
Module Name			Check Virus Total configuration		
Description			User will check the version that which version is suitable or not.		
Test Title			Test configuration		
Precondition			VT configure or not.		
Dependency			Testing API's.		
Serial no.	Test steps	Expected result	Actual result	Status fail or pass	Note
1	<ul style="list-style-type: none"><li>Open terminal</li><li>Run command</li></ul>		Waiting for actual results.	Pass	Null

## 6.2 Upload malware sample:

In this module we will upload all the samples to virus total site and try to get the reports for further processing.

**Table 6.2: Upload malware sample**

Project Name: API Based Intelligent Malware Detection					
Test Case: 2					
Test Case ID			T_id:02		
Test priority (low, medium, high)			High		
Module Name			Upload malware sample		
Description			Upload sample to Virus Total site for malware analysis.		
Test Title			Submit malware sample to VT.		
Precondition			No Task is running.		
Dependency			Sending files to VT.		
Serial no.	Test steps	Expected result	Actual result	Status fail or pass	Note
1	<ul style="list-style-type: none"><li>Open Virus Total web interface</li><li>Check results.</li></ul>	The message will be shown like “Successfully completed”.	It shows Message successfully completed analysis.	Pass	If no internet then it might not show any result.

### 6.3 Collect JSON reports:

Check the VT result. After successful malware analysis it will generate malware report in JSON form. Then collect these reports.

**Table 6.3: Collect JSON reports**

Project Name: API Based Intelligent Malware Detection					
Test Case: 3					
Test Case ID		T_id:03			
Test priority(low, high)		High			
Module Name		Collect JSON reports.			
Description		Check the VT result. After successful malware analysis it will generate malware report in JSON form. Then collect these reports.			
Test Title		Check result and collect reports.			
Precondition		VT produce result or not			
Dependency		Complete reports after analysis by VT.			
s.	Test steps	Expected result	Actual result	Status fail or pass	Note
1	<ul style="list-style-type: none"><li>• Submit files by Upload Button through GUI to VT.</li><li>• Checking for any error.</li></ul>	At the end of VT console show message “Analysis is completed”	JSON reports	Pass	If reports not get check API and check Quota Limit is exceeded.

## 6.4 Extract Features:

After uploading file and completion of malware analysis it will generate a malware JSON reports

**Table 6.4: Extract Features**

Project Name: API Based Intelligent Malware Detection					
Test Case: 4					
Test Case ID		T_id:04			
Test priority (low, medium, high)		High			
Module Name		Extract Features			
Description		After uploading file and completion of malware analysis it will generate a malware JSON reports. Extract Features from these reports			
Test Title		Features Extraction			
Precondition		Raw JSON reports.			
Despondency		Desktop App is open			
S.	Test steps	Expected result	Actual result	Status fail or pass	Note
1	<ul style="list-style-type: none"><li>• Open Desktop App</li><li>• Extract data from Raw data.</li></ul>	Return extracted features from JSON reports.	It will generate a CSV file based on these extracted features.	Pass	Check all features we need are getting or not.

## 6.5 Test and Train Model:

After the features are extracted and saved in CSV file. The next step is training model based on these extracted features.

**Table 6.5: Train Model**

Project Name: API Based Intelligent Malware Detection					
Test Case: 5					
Test Case ID		T_id:05			
Test priority (low, medium, high)		High			
Module Name		Test and Train Model			
Description		After the features are extracted and saved in CSV file. The next step is training model based on these extracted features. The model is trained on the features and the data which include Malicious and benign data.			
Test Title		Model Training on Extracted Features.			
Precondition		Raw JSON reports.			
Despondency		Desktop App is open			
S.	Test steps	Expected result	Actual result	Status fail or pass	Note
1	<ul style="list-style-type: none"><li>• Open Desktop App</li><li>• Pass CSV file</li></ul>	Based on CSV file the system will test and train model using ML.	It will test and train model based on extracted features.	Pass	

## 6.6 Use model for final result:

Use model to classify final result. The system will return final result. The result tells us the malicious and non-malicious data using this trained model.

**Table 6.6: Model for final Result**

Project Name: API Based Intelligent Malware Detection					
Test Case: 6					
Test Case ID		T_id:06			
Test priority (low, medium, high)		High			
Module Name		Use model for final result			
Description		Use model to classify final result. The system will return final result. The result tells us the malicious and non-malicious data using this trained model.			
Test Title		Use model to check type of file.			
Precondition		Desktop app running.			
Despondency		Desktop App is open			
S.	Test steps	Expected result	Actual result	Status fail or pass	Note
1	Use model to classify final result.	The system will extract the API from submitted JSON file and return the final result.	It will print message that how many files are malicious and non-malicious.	Pass	

## **6.7 Integration Testing:**

Integration testing is a level of software testing where individual units or components are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing. The tests are conducted to ensure that the components are working properly after interfacing.



# Chapter 7: Conclusion and Future Work

## 7.1. Conclusion:

In our developed model we can classify the malware according to its behavior. First it extract's the features from malware reports and generate a CSV file with extracted features. Then after the CSV file is generated we classify the API's of different malware's according to its type. Then we train our model based on these extracted API's. When all the above processes are complete then our trained model shows the type of malware based on its API.

## 7.2. Future Work:

The system required some improvements' which will help the system to work in more efficient way. The system that we have developed can only classify the malware based on API feature. So, in future we will train our model on more features which help us to identify the virus or its class.

# References

## **Dataset used in our project:**

[http://71.105.224.114:6969/torrents/VirusShare\\_00000.zip.torrent?3B9193870FF50310C54EA415C2F21274A795B76C](http://71.105.224.114:6969/torrents/VirusShare_00000.zip.torrent?3B9193870FF50310C54EA415C2F21274A795B76C)

## **Research paper 1:**

<https://peerj.com/articles/cs-285/>

## **Research paper 2:**

<https://www.sciencedirect.com/science/article/pii/S0167404821000742>

## **Research paper 3:**

<https://www.hindawi.com/journals/scn/2021/8077220/>

## **Dataset used in model systems:**

[https://github.com/ocatak/malware\\_api\\_class](https://github.com/ocatak/malware_api_class)

## **Supporting Codes we studied:**

<https://github.com/cyber-research/APTAttribution>

<https://github.com/cyber-research/APTMalware>

## **Our Project on GitHub:**

[https://github.com/AnjumShehzad7/API\\_Bases\\_Malware\\_Identification](https://github.com/AnjumShehzad7/API_Bases_Malware_Identification)