A MINI PROJECT

REPORT ON

CYBER THREAT PREDICTIVE ANALYTICS FOR IMPROVING CYBER SUPPLY CHAIN SECURITY

Submitted by,

KALIVIREDDDY ROHINI	23J45A0509
GUGULOTH MAHESHWARI	23J45A0507
DAYYALA SANDEEP	22J41A0579
MATHANGI ARAVIND	22J41A05A8

in partial fulfillment of the requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

Under the Guidance of

Dr. Pilla Srinivas

Assistant Professor, Computer Science and Engineering

COMPUTER SCIENCE AND ENGINEERING

MALLA REDDY ENGINEERING COLLEGE

(An UGC Autonomous Institution, Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Maisammaguda, Secunderabad, Telangana, India 500100

MAY -2025

MALLA REDDY ENGINEERING COLLEGE

Maisammaguda, Secunderabad, Telangana, India 500100

BONAFIDE CERTIFICATE

This is to certify that this major project work entitled "**CYBER THREAT PREDICTIVE ANALYTICS FOR IMPROVING CYBER SUPPLY CHAIN SECURITY**", submitted by **KALIVIREDDY ROHINI (23J45A0509), GUGULOTH MAHESHWARI (23J45A0507), DAYYALA SANDEEP (22J41A0579), MATHANGI ARAVIND (22J41A05A8)** to Malla Reddy Engineering College affiliated to JNTUH, Hyderabad in partial fulfilment for the award of **Bachelor of Technology** in **COMPUTER SCIENCE AND ENGINEERING** is a bonafide record project work carried out under our supervision during the academic year 2024 - 2025 and that this work has not been submitted elsewhere for a degree.

SIGNATURE

Dr. PILLA SRINIVAS

Assistant Professor, SUPERVISOR,

Computer Science and Engineering

Malla Reddy Engineering College

Secunderabad, 500 100

SIGNATURE

Dr. SARALA SANDYA RANI

Professor & HOD,

Computer Science and Engineering

Malla Reddy Engineering College

Secunderabad, 500 100

Submitted for Minor Project viva-voice examination held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

MALLA REDDY ENGINEERING COLLEGE

Maisammaguda, Secunderabad, Telangana, India 500100

DECLARATION

I hereby declare that the project titled "**CYBER THREAT PREDICTIVE ANALYTICS FOR IMPROVING CYBER SUPPLY CHAIN SECURITY**", submitted to Malla Reddy Engineering College (Autonomous) and affiliated with JNTUH, Hyderabad, in partial fulfilment of the requirements for the award of a Bachelor of Technology in COMPUTER SCIENCE AND ENGINEERING, represents my ideas in my own words. Wherever other's ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity, and I have not misrepresented, fabricated, or falsified any idea, data, fact, or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute. It is further declared that the project report or any part thereof has not been previously submitted to any University or Institute for the award of degree

		Signature's
KALIVIREDDY ROHINI	23J45A0509	_____
GUGULOTH MAHESHWARI	23J45A0507	_____
DAYYALA SANDEEP	22J41A0579	_____
MATHANGI ARAVIND	22J45A05A8	_____

Secunderabad -

500100

Date:

MALLA REDDY ENGINEERING COLLEGE

Maisammaguda, Secunderabad, Telangana, India 500100

ACKNOWLEDGEMENT

We would like to express our sincere thanks to **Dr. A. Ramaswami Reddy, professor, Principal, MREC(A)** for providing the working facilities in the college.

Our sincere thanks and gratitude to **Dr. Sarala Sandhya Rani, HOD Department of CSE, MREC(A)** for all the timely support and valuable suggestions during the period of our project.

We are thankful to our Project Coordinator **Mr. Naga Laxhman, Assistant Professor**, Department of Computer Science and Engineering, for his cooperation during the project work.

We are greatly thankful and indebted to our internal guide, **Mr.P.Srinivas,**

Professor & Head, Department of CSE, MREC(A) for his constant guidance, encouragement and moral support throughout the project.

Finally, we would also like to thank all the faculty and staff of the CSE Department who helped us directly or indirectly, parents and friends for their cooperation in completing the project work.

KALIVIREDDY ROHINI	23J45A0509
GUGULOTH MAHESHWARI	23J45A0507
DAYYALA SANDEEP	22J41A0579
MATHANGI ARAVIND	22J41A05A8

ABSTRACT

Cyber Supply Chain (CSC) is a complex system consisting of various interconnected sub-systems, making it vulnerable to threats and disruptions at multiple points. Ensuring cybersecurity in such a dynamic environment is challenging due to these inherent vulnerabilities. Cyber Threat Intelligence (CTI) plays a critical role in identifying, analyzing, and predicting both known and unknown threats by leveraging attributes such as attacker skill level, motivation, Tactics, Techniques, and Procedures (TTP), and Indicators of Compromise (IoC).

This study focuses on enhancing CSC security by integrating CTI with Machine Learning (ML) techniques to predict cyber threats effectively. The proposed system analyzes threats using the Microsoft Malware Prediction dataset, applying ML classifiers like Logistic Regression, Support Vector Machines, Random Forest, and Decision Tree. Attacks and TTPs serve as input features, while vulnerabilities and IoCs are treated as target outputs.

The results reveal that threats such as Spyware/Ransomware and Spear Phishing are highly predictable in CSC environments. Based on these findings, the study recommends control measures to counteract such threats. Future directions include refining the ML models for greater accuracy, extending CTI data coverage, and integrating real-time threat monitoring systems to improve overall CSC cybersecurity resilience.

KEY WORDS:

- Cyber Supply Chain (CSC)
- Cyber Threat Intelligence (CTI)
- Machine Learning (ML)
- Threat Prediction
- Tactics, Techniques, and Procedures (TTP)
- Indicators of Compromise (IoC)
- Logistic Regression
- Support Vector Machines (SVM)
- Random Forest
- Ransomware Detection
- Spear Phishing

TABLE OF CONTENTS

LIST OF ABBREVATIONS

CSC	Cyber Supply Chain
CTI	Cyber Threat Intelligence
ML	Machine Learning
TTP	Tactics, Techniques and Procedures
IoC	Indicator of Compromise
LR/LG	Logistic Regression
SVM	Support Vector Machine
RF	Random Forest
DT	Decision Tree

LIST OF FIGURES

CHAPTER-1

1.1 INTRODUCTION

Today's digital ecosystem is deeply interconnected, and organizations heavily depend on third-party vendors and external service providers to manage their operations. This interconnectedness has made Cyber Supply Chain (CSC) security more critical than ever for ensuring the reliability of service delivery and maintaining overall business continuity, especially within complex systems like Smart Cyber-Physical Systems (CPS).

However, due to the distributed nature of CSC systems, they are highly vulnerable to cyber threats. A single weak link in the supply chain can trigger cascading failures across the entire network. Real-world incidents like the Saudi Aramco cyberattack and the operations of the Dragonfly cyber espionage group highlight how attackers exploit vulnerabilities within the CSC environment to cause severe disruptions.

Most existing cybersecurity solutions focus only on known vulnerabilities without leveraging Cyber Threat Intelligence (CTI). CTI brings evidence-based insights into attacker behaviors, motivations, and tactics—known as TTPs (Tactics, Techniques, and Procedures)—as well as Indicators of Compromise (IoCs), helping in the early identification and mitigation of threats. In this project, we integrate CTI with Machine Learning (ML) techniques to detect and predict cyberattack patterns in CSC systems. The idea is to move from reactive to proactive cybersecurity by analyzing threats based on past intelligence and predicting future risks. This involves using classification algorithms such as Logistic Regression (LR), Support Vector Machine (SVM), Random Forest (RF), and Decision Tree (DT) on datasets like the Microsoft Malware Prediction dataset.

1.2 OBJECTIVES

Improve Cybersecurity in the Cyber Supply Chain

- Analyze and mitigate vulnerabilities across the CSC system using data-driven approaches.

Integrate Cyber Threat Intelligence (CTI) for Threat Analysis

- Use CTI properties like IoCs, TTPs, attacker motivation, and skill level for effective threat modeling and prediction.

Apply Machine Learning for Threat Prediction

- Implement ML classifiers (Logistic Regression, SVM, Random Forest, Decision Tree) to predict CSC-related cyberattacks accurately.

Develop Predictive Analytics using Real-world Datasets

- Use Microsoft Malware Prediction dataset to simulate real-world attack scenarios and train models for practical applicability.

Enable Proactive Security Measures

- Provide actionable intelligence by identifying high-risk areas in the CSC, allowing organizations to take preventive measures in advance.

CHAPTER 2

2.1 LITERATURE SURVEY

Cyber Supply Chain (CSC) security is a vital component in ensuring operational continuity and resilience of Smart **Cyber-Physical Systems (CPS).** With increasing reliance on third-party vendors and interconnected systems, CSCs are highly vulnerable to cascading cyberattacks. Real-world incidents like the **Dragonfly espionage campaign** and the Saudi Aramco power station attack illustrate the critical **impact of supply chain vulnerabilities.**

Machine Learning Approaches

Various studies have explored the integration of **Machine Learning (ML)** and **Cyber Threat Intelligence (CTI)** for enhancing CSC security:

1. **CSC Vulnerabilities and Attacks**
   - Studies have identified tactics like **software tampering and supply chain interdiction (NCSC Report, 2022).**
   - **Case-based research** reveals attackers exploit weak links in vendor ecosystems**.**
2. **ML-Based Prediction Techniques**
   - **SVM, Logistic Regression, and Random Forest** are applied for attack pattern detection (Anon, 2021).
   - **Predictive analytics** help forecast APTs and industrial espionage through training on **historical data.**
3. **CTI Frameworks and Trends**
   - **CTI** components like **Indicators of Compromise (IoCs) and Tactics, Techniques, and Procedures (TTPs)** are crucial for modeling threat behavior.
   - Studies emphasize **CTI's** role in early detection and reducing response time.

Challenges in Detection

- **Complex Supply Chains:** Security across multi-tier suppliers remains difficult.
- **Limited CTI Utilization:** Organizations underuse CTI insights in automated defense mechanisms.
- **Dynamic Threat Landscape:** Rapid evolution of attacker tactics limits static model reliability.

Conclusion

The integration of **ML and CTI enhances** the **predictive capabilities** of CSC systems. However, **real-time adaptation,** unified data standards, and **dynamic threat** modeling remain key research gaps. Future work must focus on scalable, **intelligent frameworks for proactive CSC defense**.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM:

In existing systems, cyber supply chain security primarily relies on traditional cybersecurity tools such as firewalls, intrusion detection systems, and routine security audits. These approaches focus on safeguarding individual assets rather than the supply chain as a whole, leading to gaps in addressing interdependencies. Security is often ensured through compliance with industry standards, which may not account for evolving cyber threats and zero-day vulnerabilities.

Additionally, current systems tend to emphasize post-incident response rather than proactive risk mitigation. While some incorporate basic threat detection tools, they rarely utilize advanced technologies like machine learning or threat intelligence. The lack of predictive analytics limits their ability to anticipate attacks, especially from third-party vendors or compromised software updates.

Overall, these systems offer limited visibility and control across complex, multi-tier supply chains. Their reactive nature and fragmented security measures make them inadequate in countering sophisticated cyberattacks targeting interconnected digital ecosystems.

3.2 DISADVANTAGES OF EXISTING SYSTEM:

1. **Reactive Security Measures:** Existing systems emphasize post-incident responses over proactive threat prevention, making them ineffective against emerging cyber risks.

2. **Fragmented Protection:** Security is applied individually to system components, ignoring the interconnected nature of cyber supply chains and exposing vulnerabilities in third-party relationships.

3. **Limited Use of Advanced Tools:** The absence of predictive analytics, machine learning, and real-time threat intelligence hampers early detection and mitigation of sophisticated cyberattacks.

4. **Compliance-Centric Approach:** Over-reliance on industry compliance creates a false sense of security, as standards often lag behind current threat landscapes.

5. **Complexity Management Issues:** Global and multi-tier supply chains introduce difficulty in maintaining uniform security, leading to gaps that attackers can exploit.

3.3 PROPOSED SYSTEM:

The proposed system aims to enhance cyber supply chain security using a proactive and intelligence-driven approach. It incorporates advanced technologies such as predictive analytics, machine learning, and threat intelligence to identify and mitigate potential threats before they impact the supply chain. The system consists of key modules that provide continuous monitoring, automated incident response, and real-time threat detection. Predictive analytics is employed to analyze large datasets and forecast cyber risks, enabling organizations to secure vulnerable supply chain points in advance. The system adopts a holistic strategy that covers the entire supply chain network, including third-party service providers, ensuring no component is overlooked. Threat intelligence integration provides dynamic insights into evolving threats, allowing for immediate adaptation to new attack methods. Automated mechanisms rapidly detect and contain cyber incidents, minimizing the chances of disruption. Overall, the proposed system significantly improves the security and resilience of supply chains against advanced cyber threats by focusing on prevention, real-time response, and continuous adaptation to the cyber threat landscape.

3.4 ADVANTAGES OF PROPOSED SYSYTEM

1. **Improved Context Understanding:** By utilizing deep learning models, the system captures word context and semantics more effectively, enhancing the accuracy of sentiment classification, especially in complex linguistic scenarios.

2. **Handling Language Nuances:** The system is capable of interpreting sarcasm, slang, and polysemous words, which are often misclassified by traditional approaches, ensuring more precise sentiment prediction in informal or social media text.

3. **Scalability and Automation:** Unlike rule-based systems, the proposed model automates feature extraction, making it scalable for large datasets and reducing manual intervention in model building and tuning.

4. **Robust Performance on Noisy Data:** The system is resilient to unstructured or noisy data typically found on social platforms, maintaining high performance even with spelling errors or grammatical inconsistencies.

5. **Adaptability to Evolving Language:** With support for transfer learning and fine-tuning, the model can be updated to keep pace with changing language trends, ensuring long-term accuracy and applicability across domains.

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS:

- Processor - Intel
- RAM - 4 GB
- Hard Disk - 260 GB
- Key Board - Standard Windows Keyboard
- Mouse - Two or Three Button Mouse

4.2 SOFTWARE REQUIREMENTS:

- **Operating system** : Windows 7,8 and 10 (32 and 64 bit).
- **Coding Language** : Python.
- **Designing** : Html, CSS

CHAPTER 5

5.1 SYSTEM ARCHITECTURE

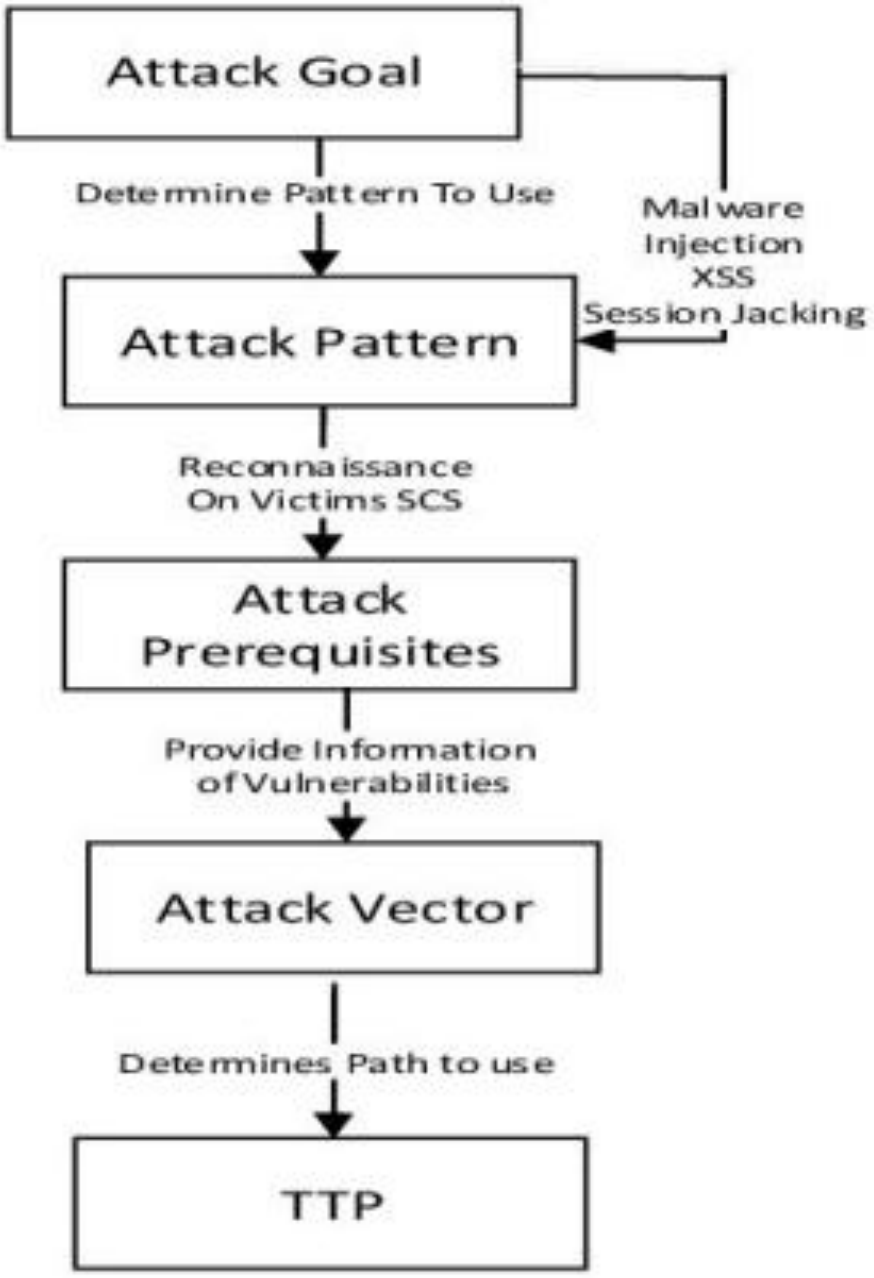

Figure 2. Supply Chain Attack Life Cycle

Fig-5.1

CHAPTER 6

6.1 SYSTEM STUDY:

FEASIBILITY STUDY

The feasibility of the project is analyzed 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

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.

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.

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.

CHAPTER 7

7.1 SYSTEM DESIGN:

INPUT DESIGN

Input Design plays a vital role in the life cycle of software development, it requires very careful attention of developers. The input design is to feed data to the application as accurate as possible. So inputs are supposed to be designed effectively so that the errors occurring while feeding are minimized. According to Software Engineering Concepts, the input forms or screens are designed to provide to have a validation control over the input limit, range and other related validations.

This system has input screens in almost all the modules. Error messages are developed to alert the user whenever he commits some mistakes and guides him in the right way so that invalid entries are not made. Let us see deeply about this under module design.

Input design is the process of converting the user created input into a computer-based format. The goal of the input design is to make the data entry logical and free from errors. The error is in the input are controlled by the input design. The application has been developed in user- friendly manner. The forms have been designed in such a way during the processing the cursor is placed in the position where must be entered. The user is also provided with in an option to select an appropriate input from various alternatives related to the field in certain cases.

Validations are required for each data entered. Whenever a user enters an erroneous data, error message is displayed and the user can move on to the subsequent pages after completing all the entries in the current page.

OUTPUT DESIGN

The Output from the computer is required to mainly create an efficient method of communication within the company primarily among the project leader and his team members, in other words, the administrator and the clients. The output of VPN is the system which allows the project leader to manage his clients in terms of creating new clients and assigning new projects to them, maintaining a record of the project validity and providing folder level access to each client on the user side depending on the projects allotted to him. After completion of a project, a new project may be assigned to the client. User authentication procedures are maintained at the initial stages itself. A new user may be created by the administrator himself or a user can himself register as a new user but the task of assigning projects and validating a new user rests with the administrator only.

The application starts running when it is executed for the first time. The server has to be started and then the internet explorer in used as the browser. The project will run on the local area network so the server machine will serve as the administrator while the other connected systems can act as the clients. The developed system is highly user friendly and can be easily understood by anyone using it even for the first time.

7.1.1 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:

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

7.1.2 USE CASE DIAGRAM :

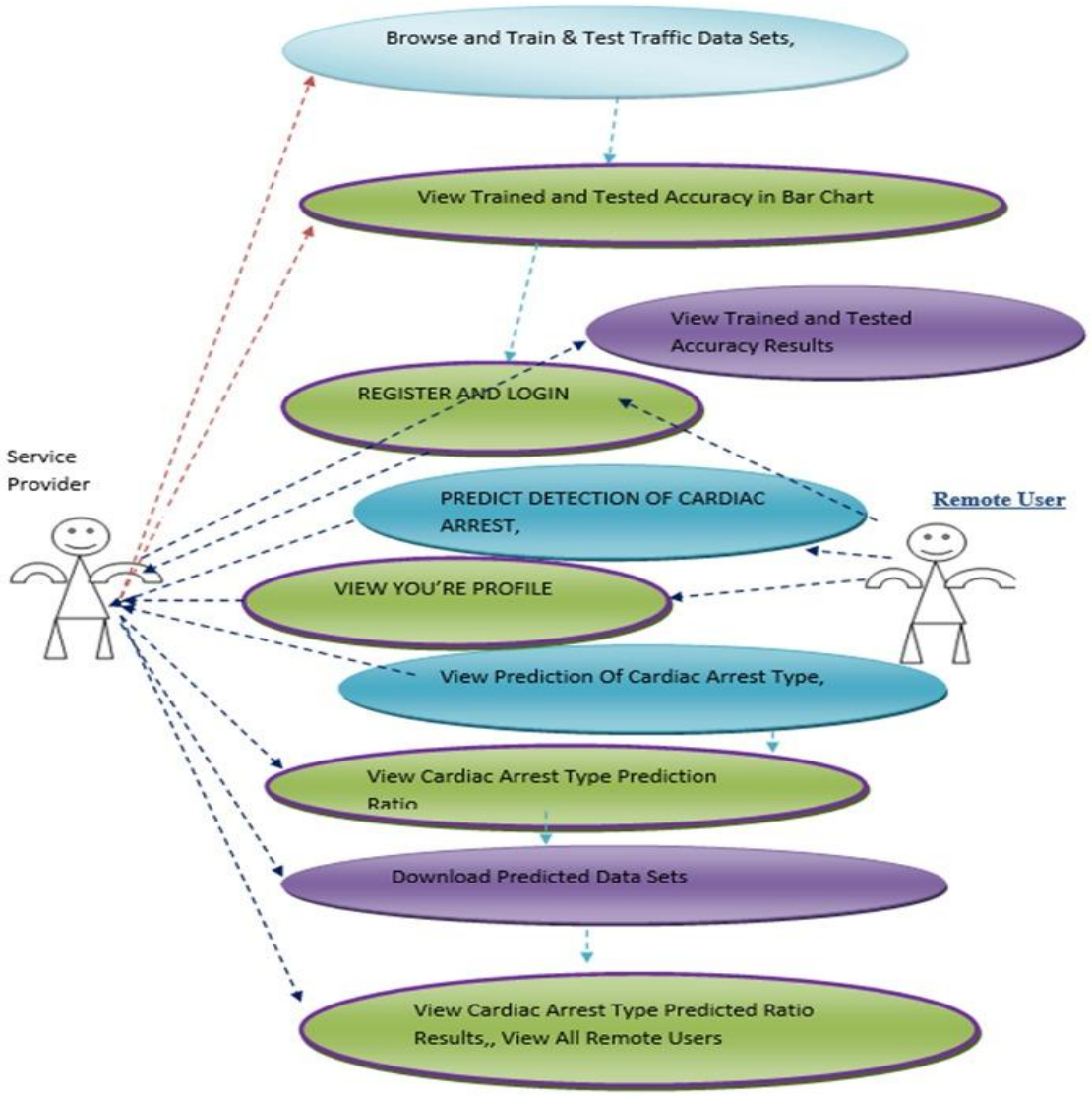

Fig-7.1.2

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral 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.

7.1.3 CLASS DIAGRAM:

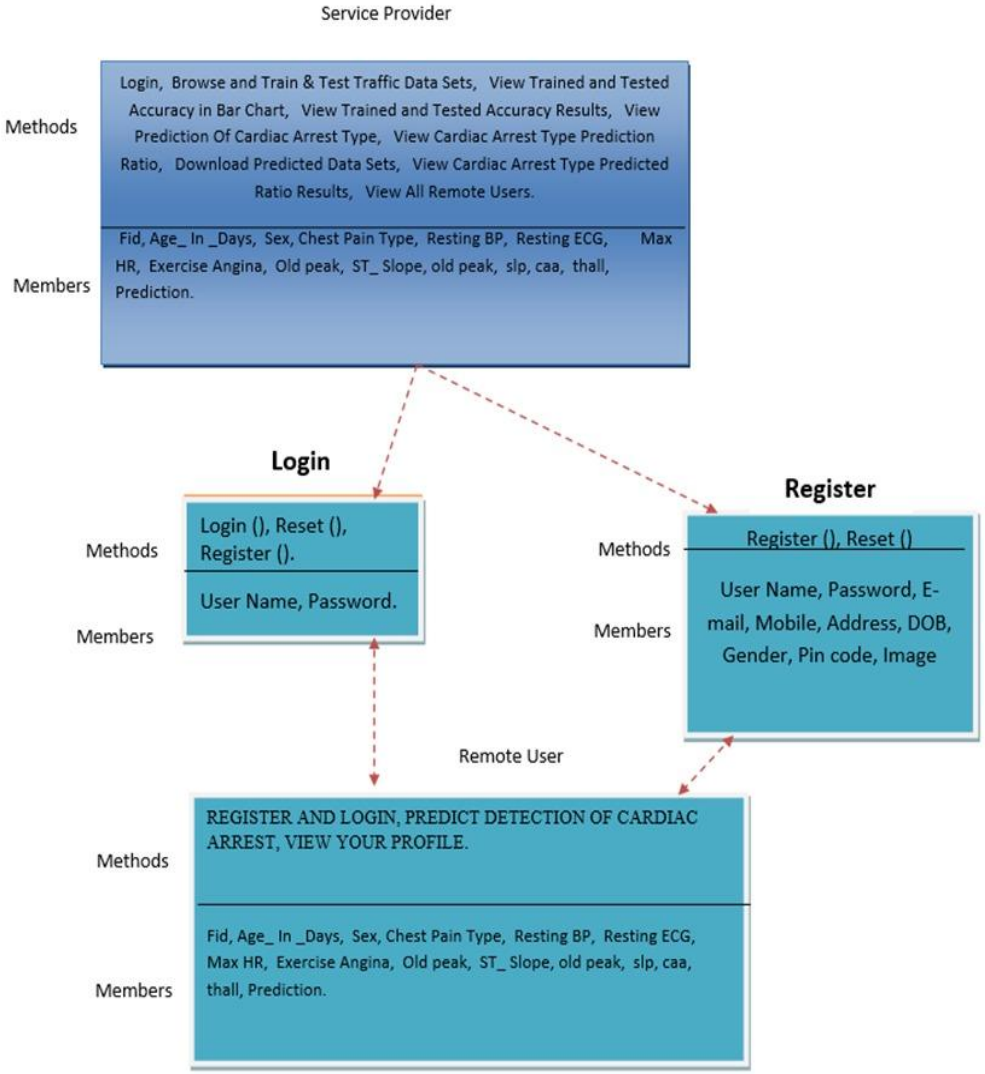

Fig-7.1.3

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. It explains which class contains information.

7.1.4 SEQUENCE DIAGRAM:

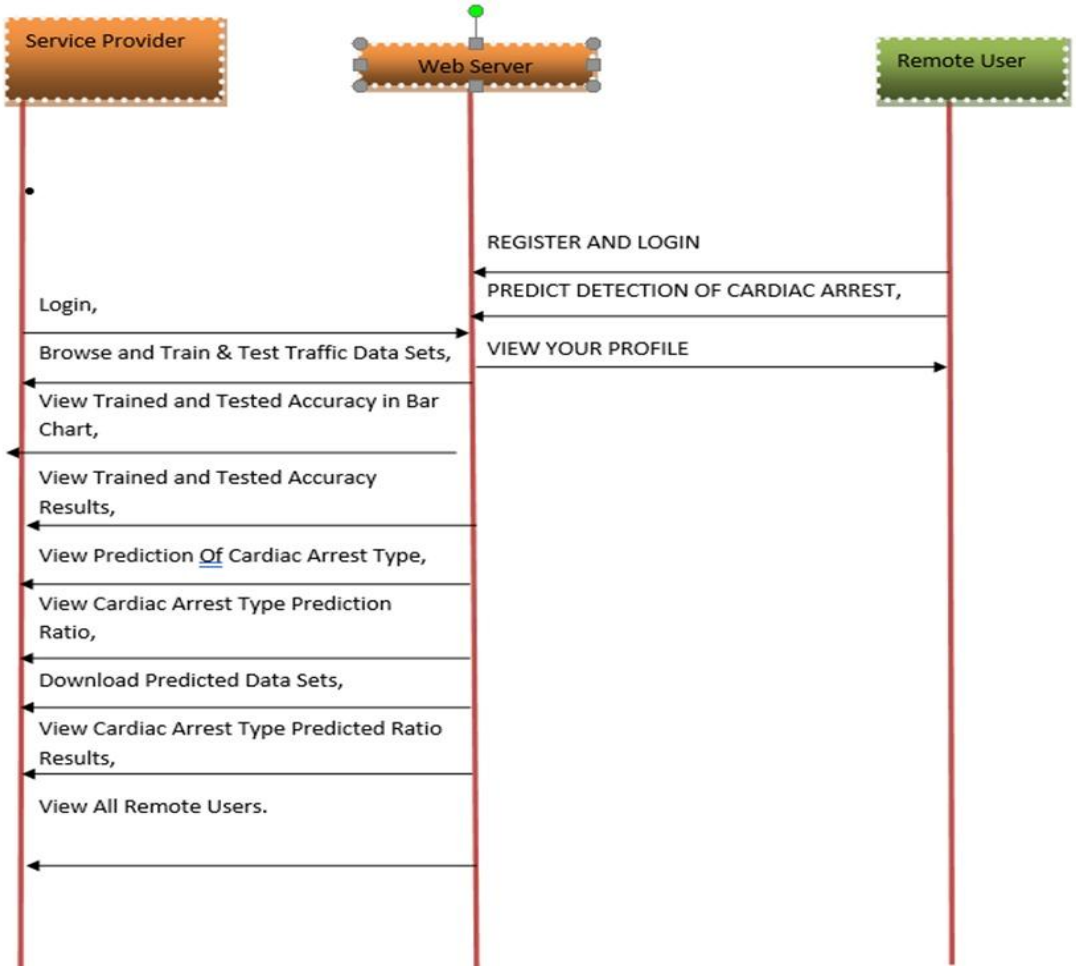

Fig-7.1.4

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.

7.1.5 DATA FLOW DIAGRAM

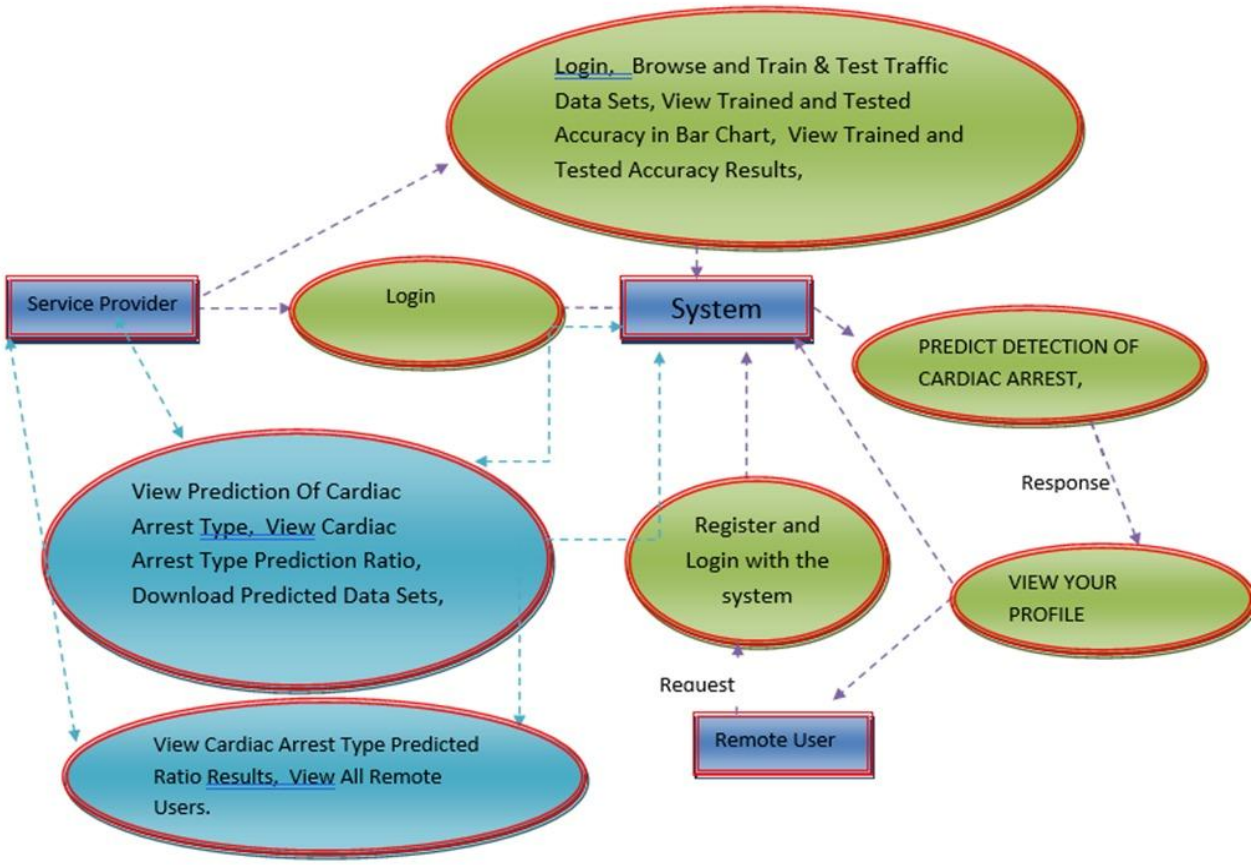

Fig-7.1.5

The system enables both service providers and remote users to log in, interact, and access cyber supply chain security analytics. Users can browse, train, and test traffic datasets to build predictive models for identifying cyber threats.

It provides accuracy results, visualizes them (such as in bar charts), and helps evaluate how well the predictions work. The system predicts cyber events — similar to predicting cardiac arrests in the original diagram — to detect potential attacks early.

Both service providers and remote users can view predictions, check vulnerability ratios, and download predicted threat datasets. It also supports monitoring by allowing users to view overall threat predictions, ratios, and activity logs from all remote users.

The workflow connects login, system access, data prediction, and result viewing, ensuring a complete threat analytics cycle that helps strengthen cyber supply chain security.

7.1.6 FLOW DIAGRAM: REMOTE USER

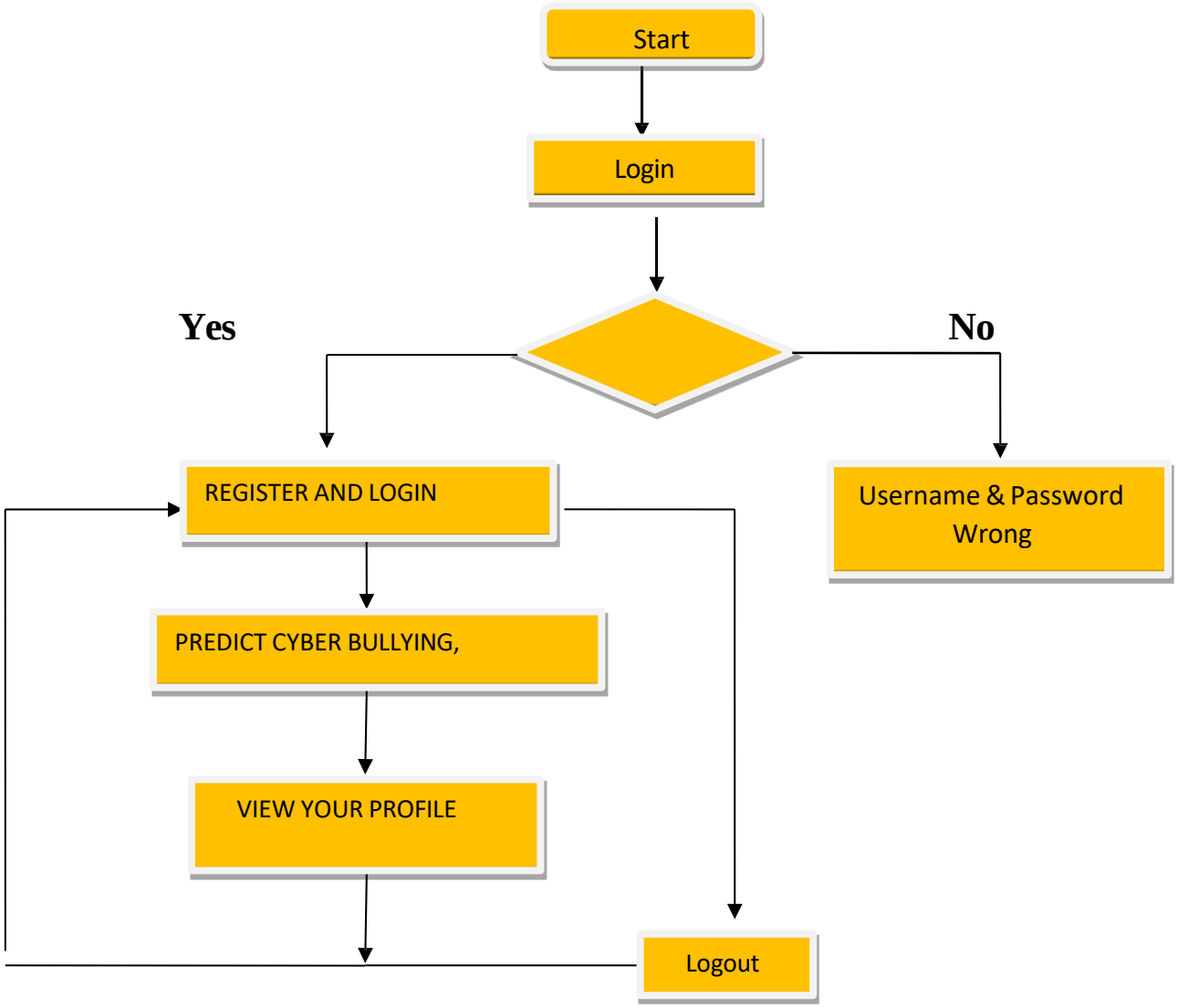

Fig-7.1.6

7.1.7 FLOW CHART : SERVICE PROVIDER

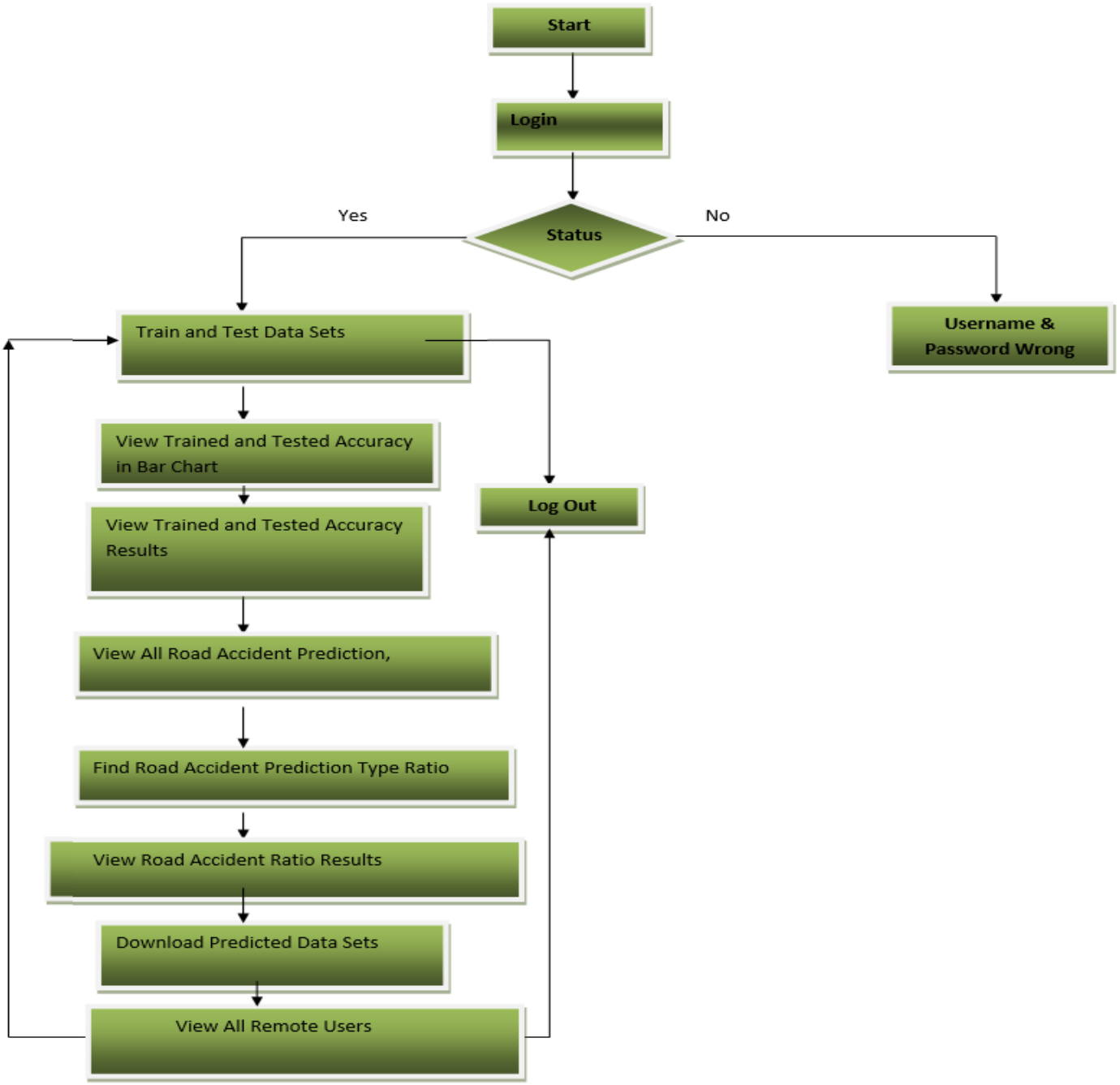

Fig: 7.1.7

7.2 SOFTWARE ENVIRONMENT:

7.2.1 PYTHON

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- 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.

7.2.2 History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

7.2.3 Python Features

Python's features include:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

- Easy-to-read: Python code is more clearly defined and visible to the eyes.

- Easy-to-maintain: Python's source code is fairly easy-to-maintain.

- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

- Interactive Mode: Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

• Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

• Databases: Python provides interfaces to all major commercial databases.

• GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

• Scalable: Python provides a better structure and support for large programs than shell scripting.

Python has a big list of good features:

• 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.

7.2.4 LIST

The list is a most versatile data type available in Python which can be written as a list of comma- separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example −

list1 = ['physics', 'chemistry', 1997, 2000];

list2 = [1, 2, 3, 4, 5];

list3 = ["a", "b", "c", "d"] Basic List Operations
Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.

Python Expression Results Description len ([1, 2, 3]) 3 Length
[1, 2, 3] + [4, 5, 6][1, 2, 3, 4, 5, 6] Concatenation

['Hi!'] * 4 ['Hi!', 'Hi!', 'Hi!', 'Hi!']Repetition 3 in [1, 2, 3] True Membership
for x in [1, 2, 3]: print x, 1 2 3 Iteration

Built-in List Functions & Methods:

Python includes the following list functions – SN Function with Description
1. cmp(list1, list2) Compares elements of both lists.
1 len (list)

Gives the total length of the list.

2 max(list)

Returns item from the list with max value.

3 min(list)

Returns item from the list with min value.

4 list(seq)

Converts a tuple into list.

Python includes following list methods SN Methods with Description
1 list.append(obj) Appends object obj to list
2 list.count(obj)

Returns count of how many times obj occurs in list

3 list. extend(seq)

Appends the contents of seq to list

4 list.index(obj)

Returns the lowest index in list that obj appears

5 list.insert(index, obj)

Inserts object obj into list at offset index

6 list.pop(obj=list[-1])

Removes and returns last object or obj from list

7 list.remove(obj) Removes object obj from list

8 list.reverse()

Reverses objects of list in place

9 list.sort([func])

Sorts objects of list, use compare function if given

7.2.5 TUPLES

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally we can put these comma-separated values between parentheses also. For example −

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5);

tup3 = "a", "b", "c", "d";

The empty tuple is written as two parentheses containing nothing − tup1 = ();

To write a tuple containing a single value you have to include a comma, even though there is only one value −

tup1 = (50,);

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

- Accessing Values in Tuples:

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example –

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5, 6, 7);

print "tup1[0]: ", tup1[0]

print "tup2[1:5]: ", tup2[1:5]

When the code is executed, it produces the following result − tup1[0]: physics

tup2[1:5]: [2, 3, 4, 5]

Updating Tuples:

Tuples are immutable which means you cannot update or change the values of tuple elements. We are able to take portions of existing tuples to create new tuples as the following example demonstrates −

```
tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');
tup3 = tup1 + tup2;
print tup3
```

When the above code is executed, it produces the following result −

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the del statement. For example: tup

= ('physics', 'chemistry', 1997, 2000);

print tup

del tup;

print "After deleting tup : "

print tup

Basic Tuples Operations:

Python Expression	ResultsDescription
len((1, 2, 3))	3 Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6) Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')Repetition
3 in (1, 2, 3)	True Membership
for x in (1, 2, 3): print x,	1 2 3 Iteration

Built-in Tuple Functions

SN	Function with Description
1	cmp(tuple1, tuple2): Compares elements of both tuples.
2	len(tuple): Gives the total length of the tuple.
3	max(tuple): Returns item from the tuple with max value.
4	min(tuple): Returns item from the tuple with min value.
5	tuple(seq): Converts a list into t

7.3.6 DICTIONARY

Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.

Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Accessing Values in Dictionary:

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example −

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

print "dict['Name']: ", dict['Name']

print "dict['Age']: ", dict['Age'] Result –

dict['Name']: Zara dict['Age']: 7

Updating Dictionary

We can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example −

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'} dict['Age'] = 8; # update existing entry

dict['School'] = "DPS School"; # Add new entry print "dict['Age']: ", dict['Age']

print "dict['School']: ", dict['School'] Result −

dict['Age']: 8 dict['School']: DPS School

Delete Dictionary Elements

We can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the del statement. Following is a simple example –

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

del dict['Name']; # remove entry with key

'Name' dict.clear(); # remove all entries

in dict

del dict ; # delete entire dictionary print "dict['Age']: ", dict['Age']

```
print "dict['School']: ", dict['School']
```

Built-in Dictionary Functions & Methods –

Python includes the following dictionary functions –

SN. Function with Description

1 cmp(dict1,dict2) Compares elements of both dict.

2 len(dict)

Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.

3 str(dict)

Produces a printable string representation of a dictionary

4 type(variable)

Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Python includes following dictionary methods –

SN. Methods with Description

1 dict.clear():Removes all elements of dictionary dict

2 dict. Copy():Returns a shallow copy of dictionary dict

3 dict.fromkeys():Create a new dictionary with keys from seq and values set to value.

4 dict.get(key, default=None):For key key, returns value or default if key not in dictionary

5 dict.has_key(key):Returns true if key in dictionary dict, false otherwise

6 dict.items():Returns a list of dict's (key, value) tuple pairs

7 dict.keys():Returns list of dictionary dict's keys

8 dict.setdefault(key, default=None):Similar to get(), but will set dict[key]=default if key is not already in dict

9 dict.update(dict2):Adds dictionary dict2's key-values pairs to dict

10 dict.values():Returns list of dictionary dict's values

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions.

Defining a Function

Simple rules to define a function in Python.

- Function blocks begin with the keyword def followed by the function name and parentheses (( )).

- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.

- The first statement of a function can be an optional statement - the documentation string of the function or docstring.

- The code block within every function starts with a colon (:) and is indented.

- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

def functionname(parameters):

"function_docstring" function_suite

return [expression]

Calling a Function

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt. Following is the example to call printme() function −

Function definition is here def printme(str):
"This prints a passed string into this function" print str

return;

Now you can call printme function printme("I'm first call to user defined function!")

printme("Again second call to the same function")

When the above code is executed, it produces the following result − I'm first call to user defined function!
Again second call to the same function Function Arguments
You can call a function by using the following types of formal arguments:

- Required arguments
- Keyword arguments
- Default arguments
- Variable-length arguments

Scope of Variables

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python −

Global variables Local variables Global vs. Local variables
Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope. Following is a simple example −

total = 0; # This is global variable. # Function definition is here
def sum(arg1, arg2):

# Add both the parameters and return them." total = arg1 + arg2; # Here total is local variable. print "Inside the function local total : ", total return total;
sum(10, 20);

print "Outside the function global total : ", total Result −

Inside the function local total : 30 Outside the function global total : 0
A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

Example:

The Python code for a module named aname normally resides in a file named aname.py. Here's an example of a simple module, support.py

def print_func(par):

print "Hello : ", par return
The import Statement

The import has the following syntax:

import module1[, module2[,... moduleN]

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches before importing a module. For example, to import the module support.py, you need to put the following command at the top of the script –

A module is loaded only once, regardless of the number of times it is imported. This prevents the module execution from happening over and over again if multiple imports occur.

Packages in Python

A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and sub packages and sub-sub packages.

Consider a file Pots.py available in Phone directory. This file has following line of source

code −

```
def Pots():

print "I'm Pots Phone"
```

Similar way, we have another two files having different functions with the same name as

Above -

- Phone/Isdn.py file having function Isdn()
- Phone/G3.py file having function G3()

Now, create one more file __init__.py in Phone directory −

- Phone/__init__.py

To make all of your functions available when you've imported Phone,to put explicit import statements in __init__.py as follows –

```
from Pots import Pots from Isdn import Isdn from G3 import G3
```

After you add these lines to __init__.py, you have all of these classes available when you import the Phone package.

```
# Now import your Phone Package.
import Phone
Phone.Pots()
Phone.Isdn()
Phone.G3()
RESULT:
I'm Pots
Phone I'm
3G Phone
I'm ISDN
Phone
```

In the above example, we have taken example of a single functions in each file, but you can keep multiple functions in your files. You can also define different Python classes in those files and then you can create your packages out of those classes.

This chapter covers all the basic I/O functions available in Python. Printing to the Screen
The simplest way to produce output is using the print statement where you can pass zero or more expressions separated by commas. This function converts the expressions you pass into a string and writes the result to standard output as follows −

print "Python is really a great language,", "isn't it?" Result:
Python is really a great language, isn't it? Reading Keyboard Input
Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are −

- raw_input
- input

The raw_input Function

The raw_input([prompt]) function reads one line from standard input and returns it as a string (removing the trailing newline).

```
str = raw_input("Enter your input: ");
print "Received input is : ", str
```

This prompts you to enter any string and it would display same string on the screen. When I typed "Hello Python!", its output is like this −

Enter your input: Hello Python Received input is : Hello Python The input Function
The input([prompt]) function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

```
str = input("Enter your input: "); print "Received input is : ", str
```

This would produce the following result against the entered input − Enter your input: [x*5 for x in range(2,10,2)]
Recieved input is : [10, 20, 30, 40] Opening and Closing Files
Until now, you have been reading and writing to the standard input and output. Now, we will see how to use actual data files.

Python provides basic functions and methods necessary to manipulate files by default. You can do most of the file manipulation using a file object.

The open Function

Before you can read or write a file, you have to open it using Python's built-in open() function. This function creates a file object, which would be utilized to call other support methods associated with it.

Syntax

file object = open(file_name [, access_mode][, buffering]) Here are parameter details:

- file_name: The file_name argument is a string value that contains the name of the file that you want to access.

- access_mode: The access_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is optional parameter and the default file access mode is read (r).

- buffering: If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default(default behavior).

Here is a list of the different modes of opening a file −

Modes Description

r Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.

rb Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.

r+ Opens a file for both reading and writing. The file pointer placed at the beginning of the file.

rb+Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.

w Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

wb Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

w+ Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

wb+

Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.

a Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

ab Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.

a+ Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a

new file for reading and writing.

ab+ Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

The file Object Attributes

Once a file is opened and you have one file object, you can get various information related to that file.

Here is a list of all attributes related to file object:

Attribute	Description
file.closed	Returns true if file is closed, false otherwise.
file.mode	Returns access mode with which file was opened.
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

Example

```
# Open a file
fo = open("foo.txt", "wb")
print "Name of the file: ", fo.name
print "Closed or not : ", fo.closed print
"Opening mode : ", fo.mode print
"Softspace flag : ", fo.softspace This
produces the following result − Name
of the file: foo.txt
Closed or not : False
Opening mode : wb
Softspace flag : 0
```

The close() Method

The close() method of a file object flushes any unwritten information and closes the file object, after which no more writing can be done.Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.

Syntax

fileObject.close()

;

Example

 # Open a file

fo = open("foo.txt", "wb")

print "Name of the file: ", fo.name # Close opend file

fo.close() Result −

Name of the file: foo.txt Reading and Writing Files

The file object provides a set of access methods to make our lives easier. We would see how to use read() and write() methods to read and write files.

The write() Method

The write() method writes any string to an open file. It is important to note that Python strings can have binary data and not just text.The write() method does not add a newline character ('\n') to the end of the string Syntax

fileObject.write(string);

Here, passed parameter is the content to be written into the opened file. Example # Open a

file

fo = open("foo.txt", "wb")

fo.write("Python is a great language.\nYeah its great!!\n");

Close opend file fo.close()

The above method would create foo.txt file and would write given content in that file and finally it would close that file. If you would open this file, it would have following content.

Python is a great language. Yeah its great!!
The read() Method

The read() method reads a string from an open file. It is important to note that Python strings can have binary data. apart from text data.

Syntax fileObject.read([count]);
Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.

Example

Let's take a file foo.txt, which we created

above. # Open a file

fo = open("foo.txt",

"r+") str = fo.read(10);

print "Read String is : ", str # Close opend file fo.close()

This produces the following result − Read String is : Python is

Create a directory "test" os.mkdir("test")

The chdir() Method

You can use the chdir() method to change the current directory. The chdir() method takes an argument, which is the name of the directory that you want to make the current directory.

Syntax os.chdir("newdir") Example

Following is the example to go into "/home/newdir" directory − #!/usr/bin/python

import os

Changing a directory to "/home/newdir" os.chdir("/home/newdir")

The getcwd() Method

The getcwd() method displays the current working directory.

Syntax os.getcwd() Example

Following is the example to give current directory − import os

CHAPTER 8

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.

8.1 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 testing

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.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted.
Invalid Input : identified classes of invalid input must be rejected.
Functions : identified functions must be exercised.
Output : identified classes of application outputs must be exercised.
Systems/Procedures: interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

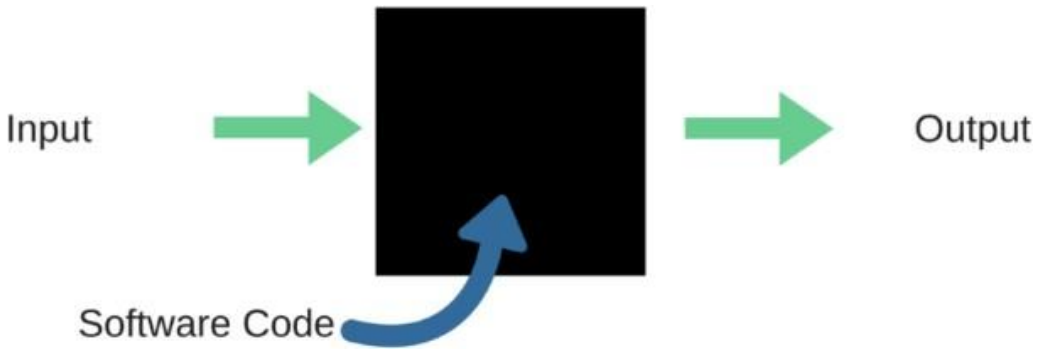

Fig: 8.1.1

When applied to machine learning models, black box testing would mean testing machine learning models without knowing the internal details such as features of the machine learning model, the algorithm used to create the model etc. The challenge, however, is to verify the test outcome against the expected values that are known beforehand.

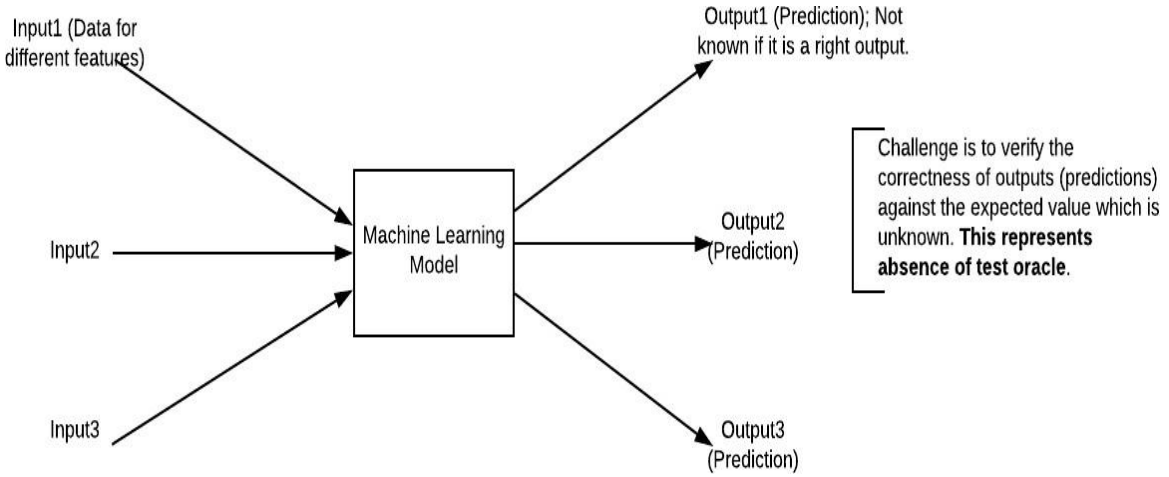

Fig : 8.1.2

Unit Testing :

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

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.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects. The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

IMPLEMENTATION/SCREENSHORT:

To enhance cyber supply chain security, several machine learning classifiers can be employed to detect and predict cyber threats. Here's how each of these classifiers might work within the context of your project:

1. **Naive Bayes**

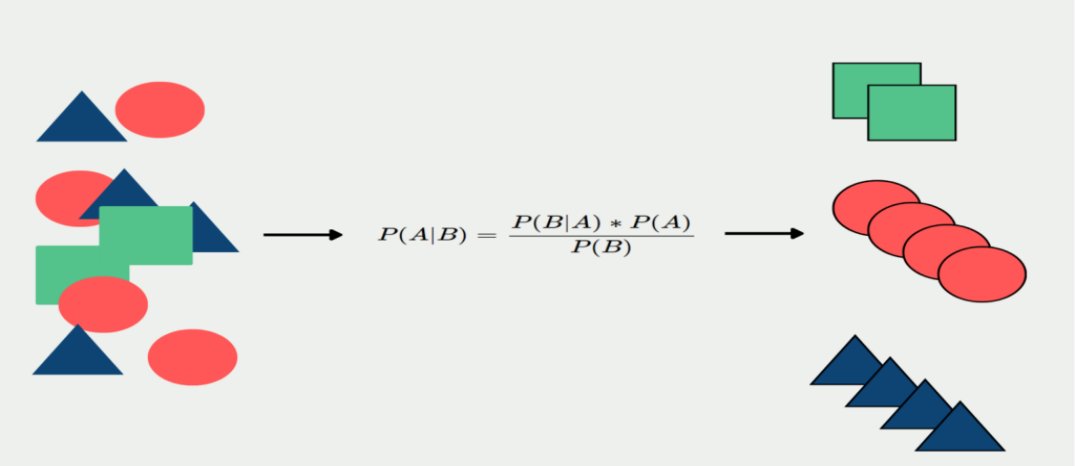

Fig : 8.1.3

-> **How It Works:** Naive Bayes is a probabilistic classifier based on Bayes' theorem. It assumes that the features used to classify an instance are conditionally independent given the class label.

-> **Application in the Project:** Naive Bayes can be used to classify potential threats by analyzing patterns in historical data. For example, it can predict whether a particular communication within the supply chain is benign or malicious based on prior occurrences of similar patterns.

2. **Support Vector Machine (SVM)**

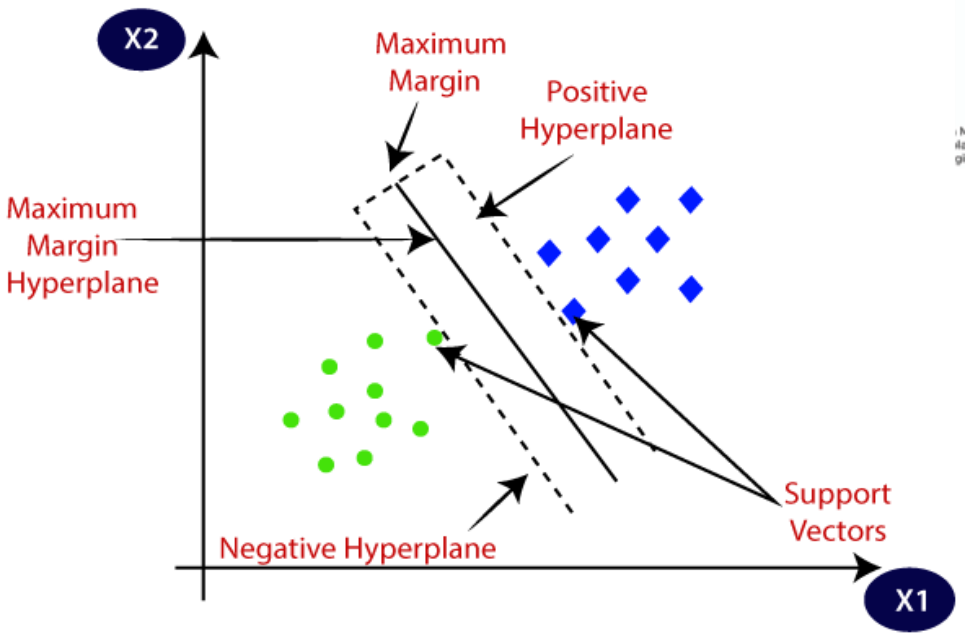

Fig : 8.1.4

-> **How It Works:** SVM is a supervised learning algorithm that finds the optimal hyperplane to separate data points into different classes. It works well with high-dimensional data and is effective for binary classification problems.

-> **Application in the Project:** SVM can be used to detect anomalies in supply chain communications or transactions. By mapping data points into a high-dimensional space, SVM can distinguish between normal and suspicious activities, helping to identify potential cyber threats.

3. **Logistic Regression**

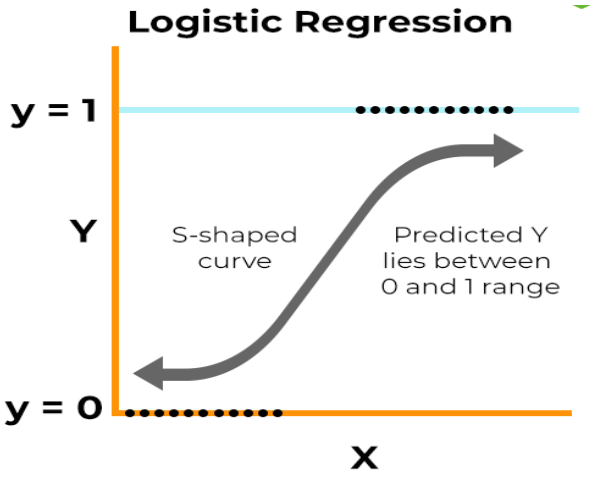

Fig : 8.1.5

-> **How It Works:** Logistic regression models the probability that a given input belongs to a certain class. It is particularly useful for binary classification tasks where the outcome is either a 0 or 1.

-> **Application in the Project:** Logistic regression can be employed to predict the likelihood of a cyberattack occurring based on specific indicators or features within the supply chain data, such as unusual login times or unexpected file transfers.

4. **Decision Tree Classifier**

-> **How It Works:** A decision tree classifier splits the data into subsets based on the value of input features, creating a tree-like model of decisions. It is easy to interpret and can handle both categorical and numerical data.

-> **Application in the Project:** Decision trees can be used to classify different types of threats based on various attributes, such as the origin of a communication, the type of file being transferred, or the behavior of users within the supply chain.

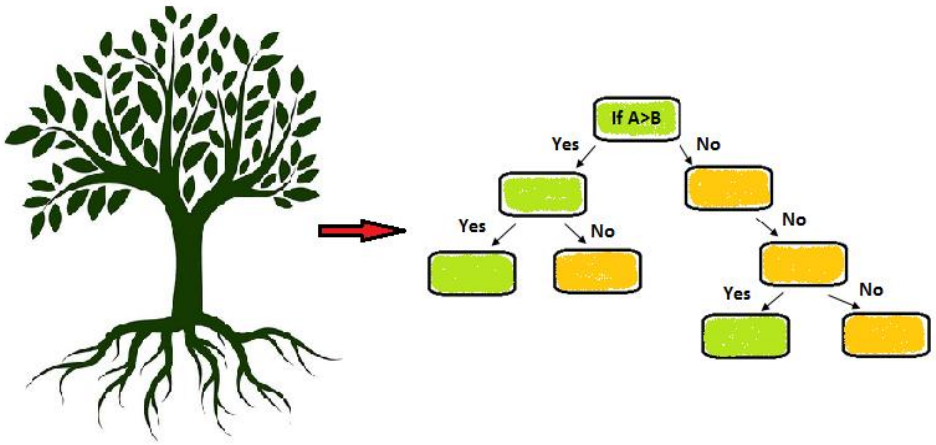

Fig : 8.1.6

5. **Stochastic Gradient Descent (SGD) Classifier**

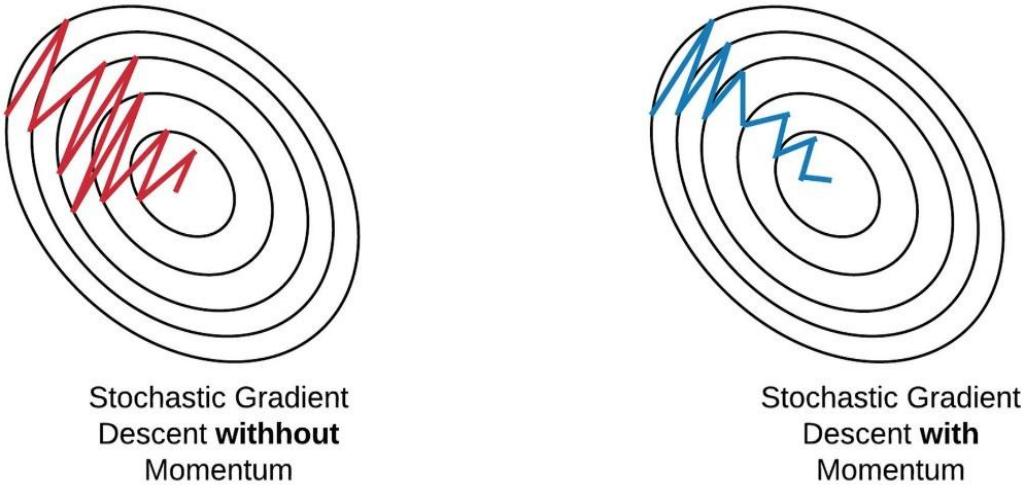

Fig : 8.1.7

-> How It Works: The SGD classifier is a linear model that uses stochastic gradient descent for optimization. It is efficient for large-scale datasets and can be applied to different loss functions like hinge loss (SVM) or log loss (logistic regression).

-> Application in the Project: SGD can be used to quickly classify large volumes of data within the supply chain, such as network traffic or transaction logs, to identify potential threats in real-time.

6. **K-Nearest Neighbours (KNN)**

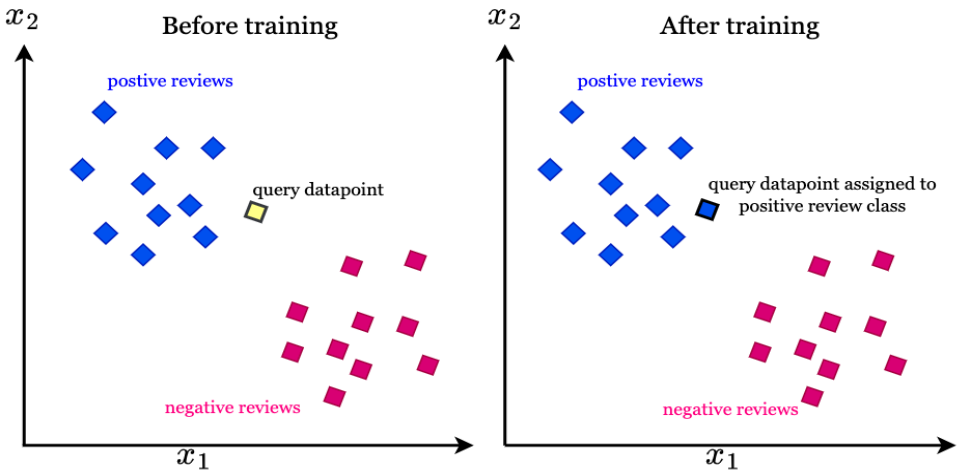

Fig : 8.1.8

-> How It Works: KNN is a non-parametric algorithm that classifies instances based on the majority class of the k-nearest neighbours in the feature space. It is simple and effective for small to medium-sized datasets.

-> Application in the Project: KNN can be used to detect anomalies by comparing new data points with historical data. For instance, if a new transaction or communication is significantly different from its neighbours, it could be flagged as suspicious.

7. **Random Forest Classifier**

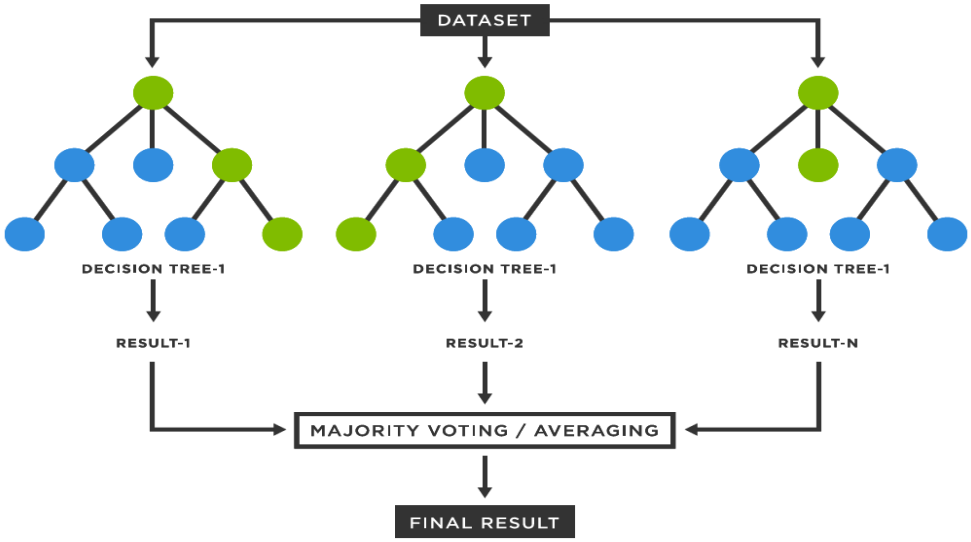

Fig : 8.1.9

-> How It Works: Random Forest is an ensemble learning method that combines multiple decision trees to improve classification accuracy and reduce overfitting. Each tree in the forest votes on the class label, and the majority vote is taken as the final prediction.

-> Application in the Project: Random Forest can be used to identify complex patterns in supply chain data that might indicate a cyber threat. Its robustness to overfitting and ability to handle large datasets make it suitable for analyzing diverse and complex data sources within the supply chain.

Implementation Steps:

1. Data Collection: Gather historical data on supply chain activities, including network logs, transactions, user behaviours, etc. This data will serve as the input features for training the models.

2. Feature Engineering: Select and engineer features that are indicative of cyber threats. These could include timestamps, IP addresses, file types, user roles, and other relevant attributes.

3. Model Training: Train each of the classifiers (Naive Bayes, SVM, Logistic Regression, etc.) on labeled data, where the outcomes (e.g., benign or malicious) are known.

4. Model Evaluation: Evaluate the performance of each model using metrics like accuracy, precision, recall, and F1-score. This will help determine which classifier is most effective for your specific use case.

5. Model Deployment: Deploy the best-performing models in your cyber supply chain security system. The models can be integrated into real-time monitoring systems to continuously classify new data and identify potential threats.

6. Continuous Improvement: Regularly update and retrain the models with new data to adapt to evolving threats and ensure the system remains effective over time.

By using these classifiers, the proposed system can enhance the detection and prediction of cyber threats within the supply chain, ultimately improving overall security.

CHAPTER 9

OUTPUT SCREEN

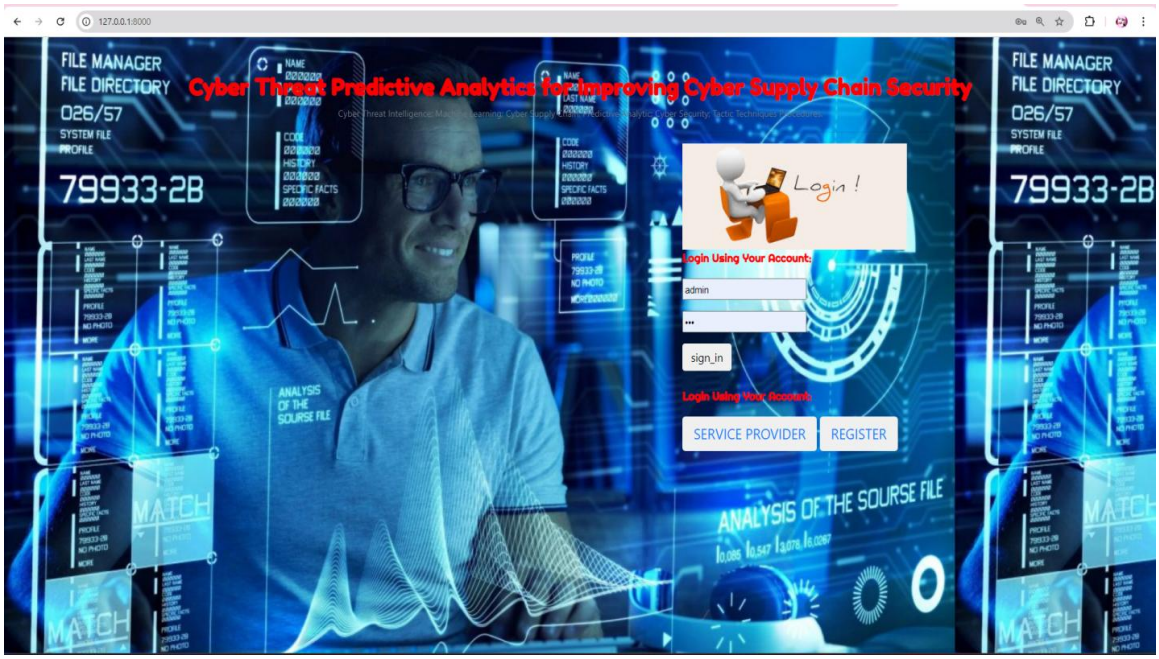

Fig- 9.1.1

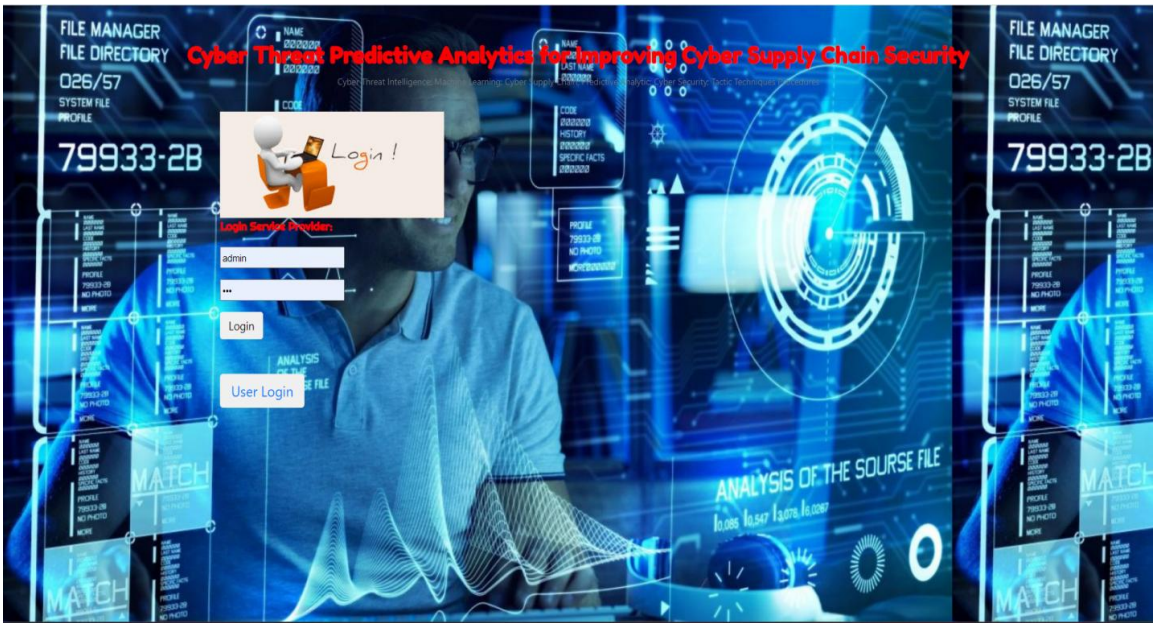

Fig- 9.1.2

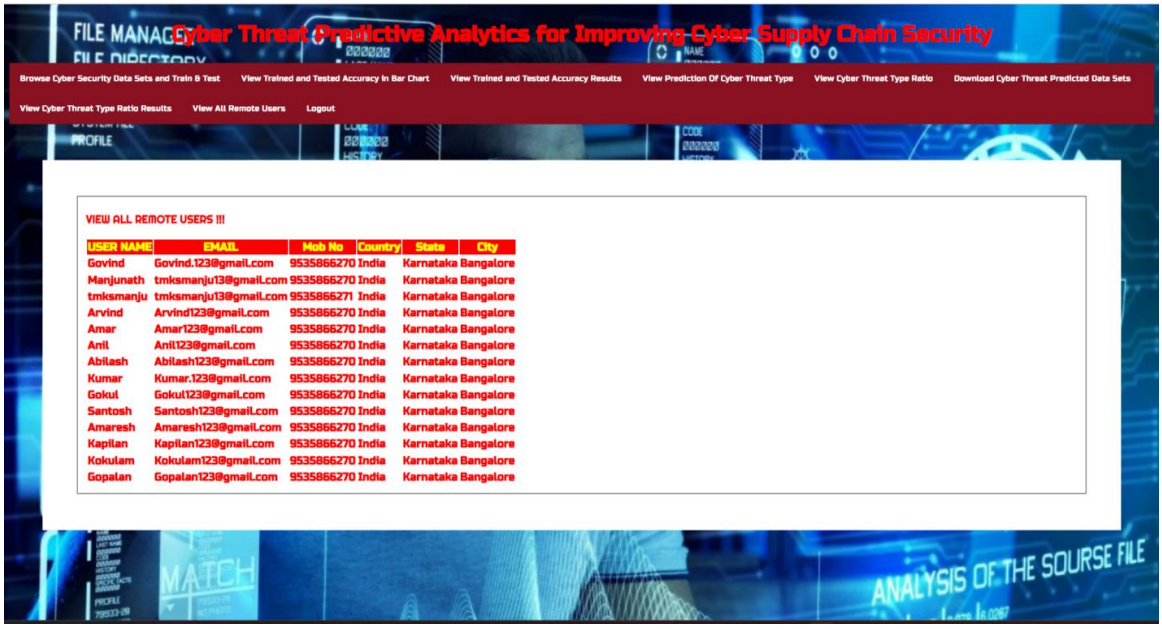

Fig- 9.1.3

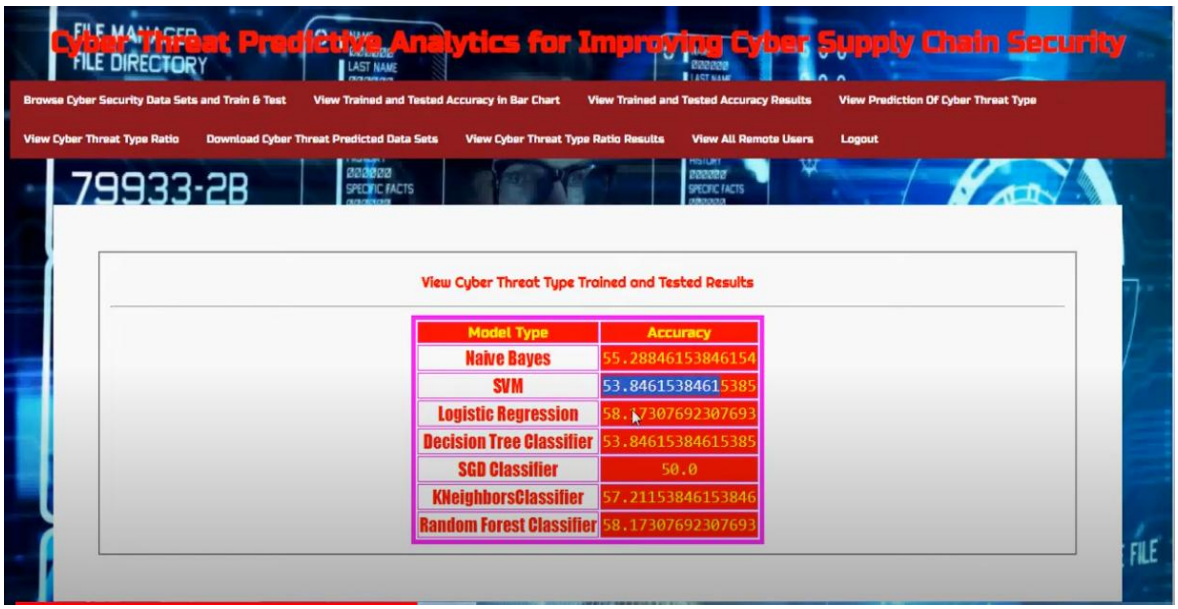

Fig- 9.1.4

Fig- 9.1.5

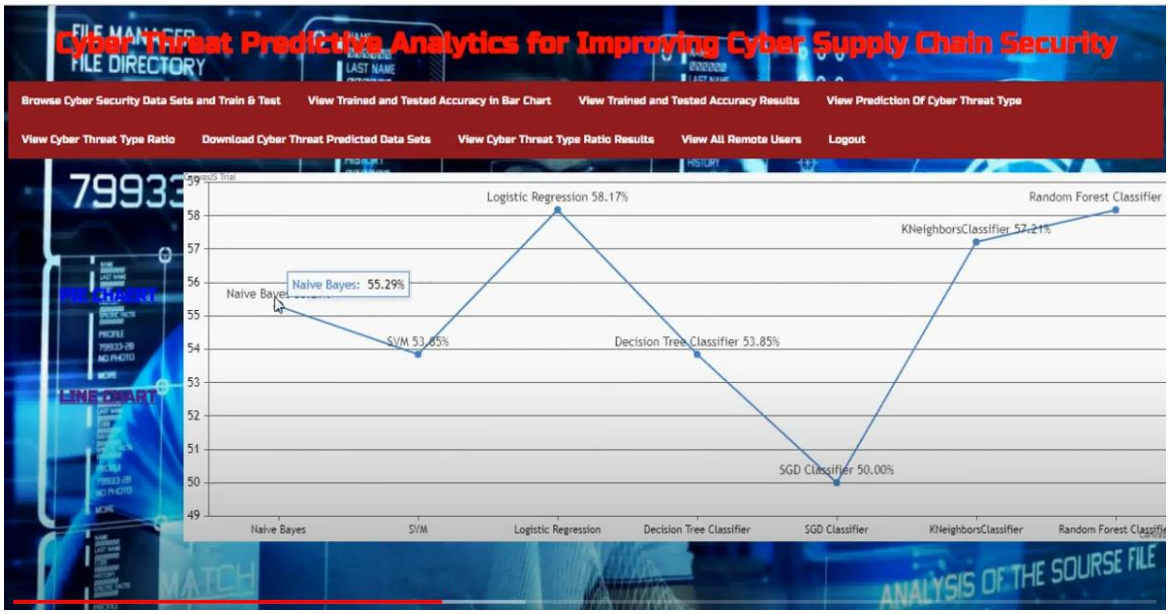

Fig- 9.1.6

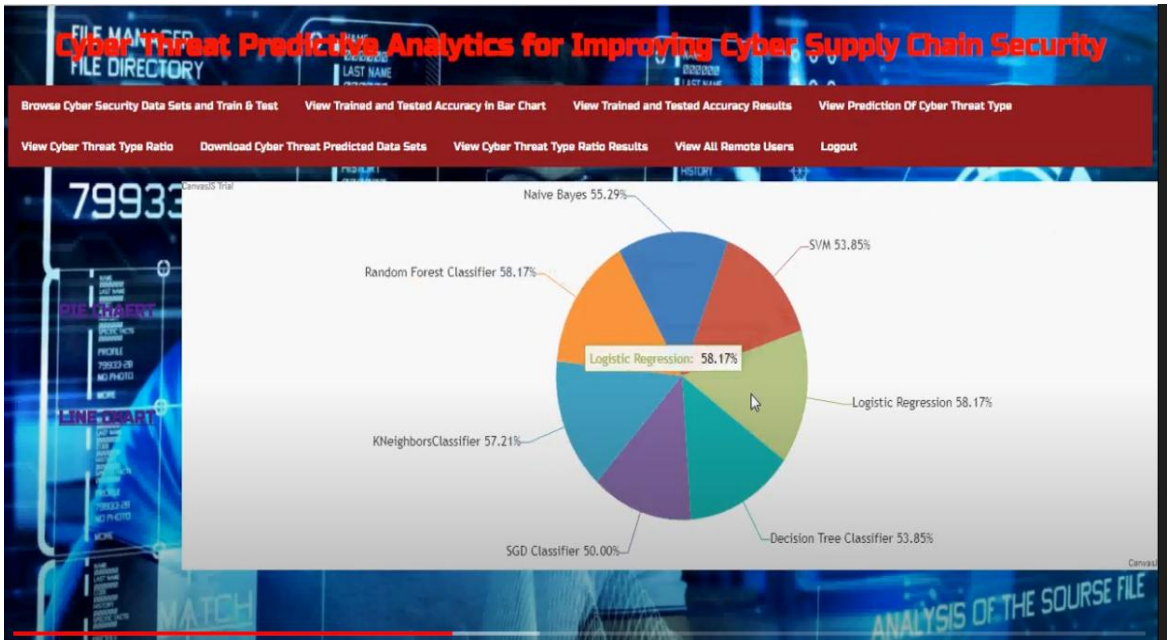

Fig- 9.1.7

Fig- 9.1.8

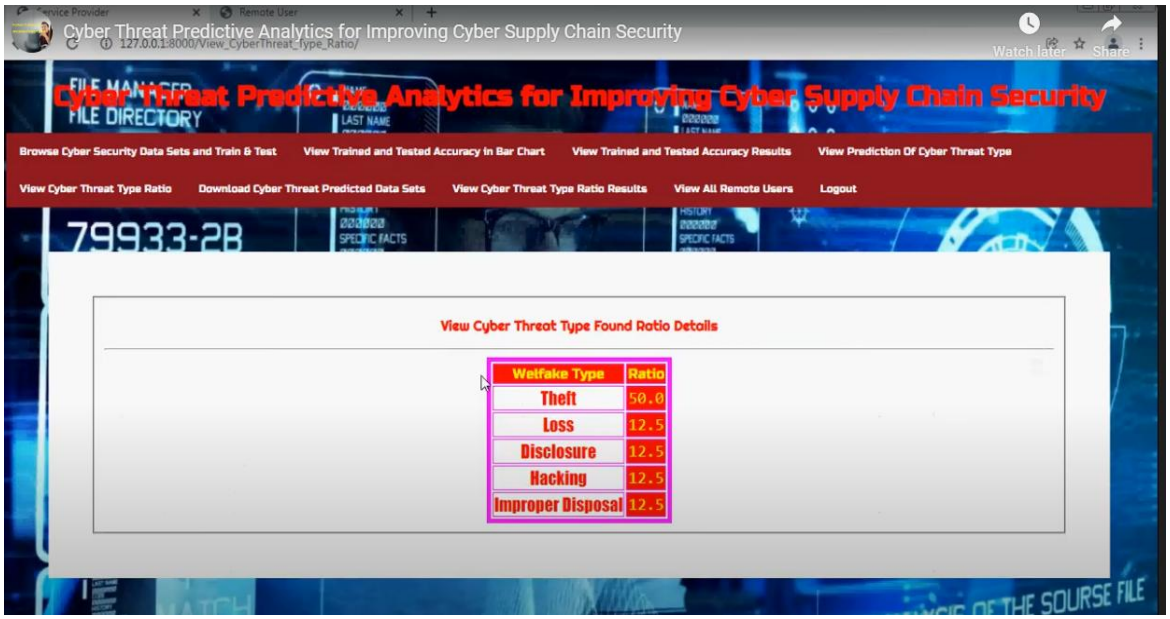

Fig- 9.1.9

Fig- 9.1.10

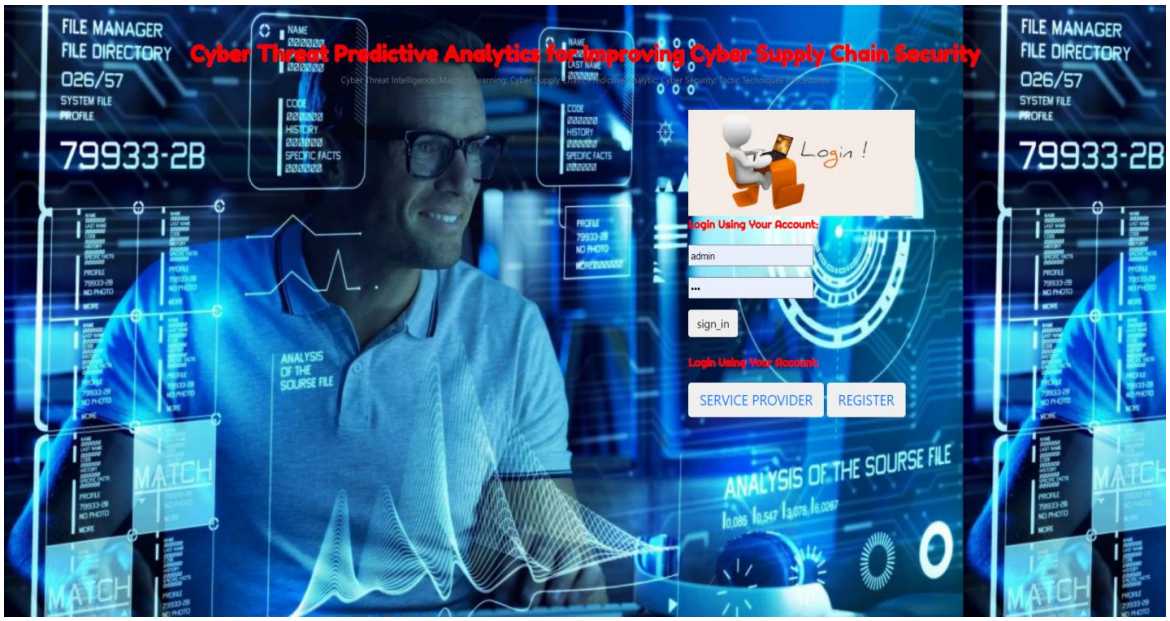

Fig- 9.1.11

Fig- 9.1.12

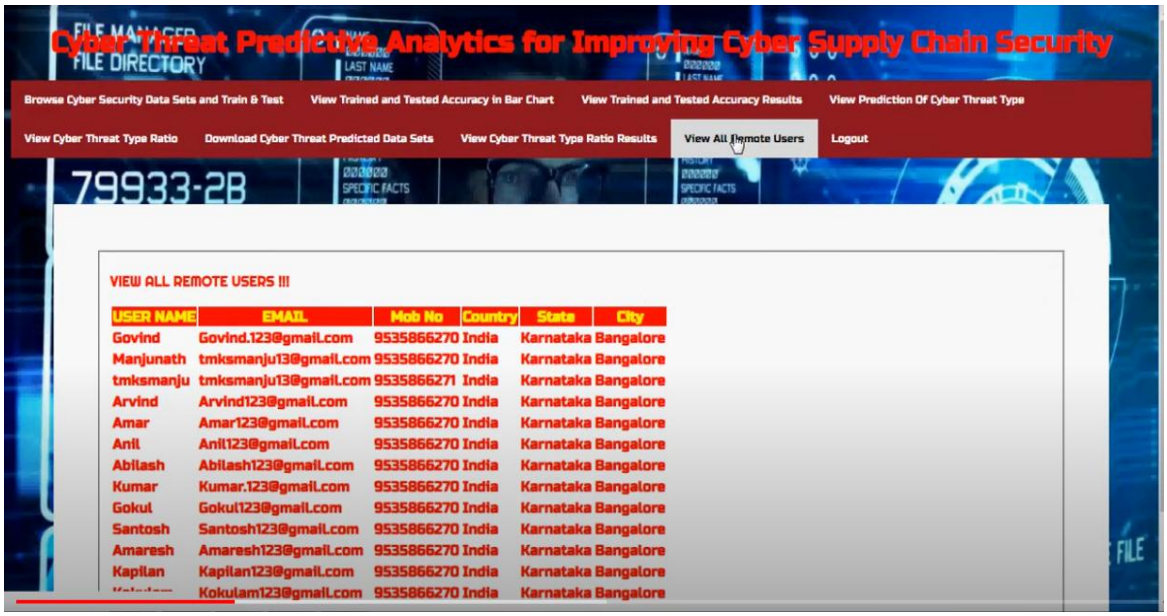

Fig- 9.1.13

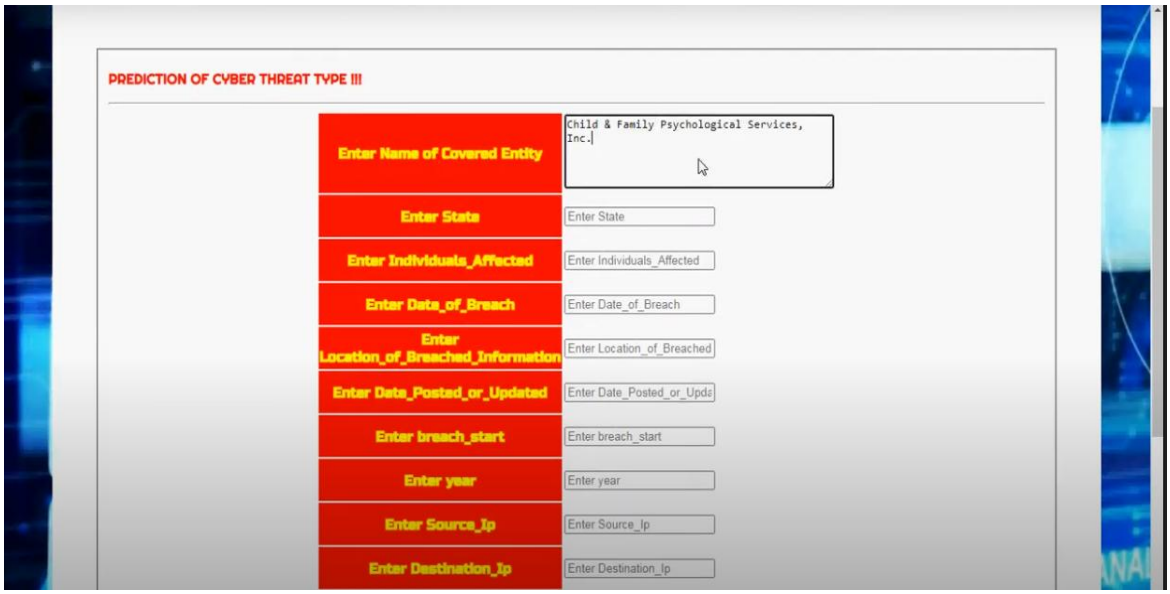

Fig- 9.1.14

CHAPTER-10

CONCLUSION

In conclusion, enhancing **Cyber supply chain security** through the integration of advanced machine learning classifiers offers a proactive and comprehensive approach to mitigating the ever evolving threats that organizations face today. By employing models such as **Naive Bayes, SVM, Logistic Regression, Decision Trees, SGD, KNN, and Random Forest,** the system can effectively identify, predict, and respond to potential cyber threats before they can cause significant damage. These classifiers not only improve the accuracy of threat detection but also **enable real-time monitoring and rapid response**, thereby reducing the risk of disruptions in critical supply chain operations. The implementation of such a system ensures that organizations are better equipped to safeguard their assets, maintain business continuity, and **adapt to new challenges in the cyber security landscape**. Ultimately, this integrated, intelligence-driven approach represents a significant advancement in the **pursuit of robust cyber supply chain security.**

CHAPTER 11

REFERENCES

1. National Cyber Security Centre. (2018). Example of Supply Chain Attacks.
2. A. Yeboah-Ofori and S. Islam, ‘‘Cyber security threat modelling for supply chain organizational environments,’’ MDPI. Future Internet, vol. 11, no. 3, p. 63, Mar. 2019.
3. B. Woods and A. Bochman, ‘‘Supply chain in the software era,’’ in Scowcroft Center for Strategic and Security. Washington, DC, USA: Atlantic Council, May 2018.
4. Exploring the Opportunities and Limitations of Current Threat Intelligence Platforms, Version 1, ENISA, Dec. 2017.
5. C. Doerr, TU Delft CTI Labs. (2018). Cyber Threat Intelligences Standards—A High Level Overview.
6. Research Prediction. (2019). Microsoft Malware Prediction.
7. A.Yeboah-Ofori and F. Katsriku, ‘‘Cybercrime and risks for cyber physical systems,’’ Int. J. Cyber-Secur. Digit. Forensics, vol. 8, no. 1, pp. 43–57, 2019.
8. CAPEC-437, Supply Chain. (Oct. 2018). Common Attack Pattern Enumeration and Classification: Domain of Attack.
9. Open Web Application Security Project (OWASP). (2017). The Ten Most Critical Application Security Risks, Creative Commons Attribution-Share Alike 4.0 International License.
10. US-Cert. (2020). Building Security in Software & Supply Chain Assurance