**Automated Emerging Cyber Threat Identification**

A Mini Project Report submitted to

JNTU Hyderabad in partial fulfillment

of the requirements for the award of the degree

**BACHELOR OF TECHNOLOGY**

In

**COMPUTER SCIENCE AND ENGINEERINRG**

***Submitted by***

#### RAMASWAMI SHERUKULA MOUNIKA 21RG1A05B6

#### NIMMAKANTI DIVYA 21RG1A05A6

#### MADDIKUNTA AKHILA 21RG1A0596

#### THATIPAMULA POOJITHA 21RG1A05C5

*Under the Guidance of*

**Mr. Prasanth Kumar Kunda**

*Assistant Professor*



***DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING***

**MALLA REDDY COLLEGE OF ENGINEERING FOR WOMEN**

**An UGC Autonomous Institution**

*Approved by AICTE New Delhi and Affiliated to JNTUH*

*Maisammaguda, Medchal (Dist), Hyderabad -500100, Telangana.*

*OCTOBER 2024*

i

***DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING***

**MALLA REDDY COLLEGE OF ENGINEERING FOR WOMEN**

**An UGC Autonomous Institution**

*Approved by AICTE New Delhi and Affiliated to JNTUH*

*Maisammaguda, Medchal (Dist), Hyderabad -500100, Telangana.*

*OCTOBER 2024*



***CERTIFICATE***

This is to certify that the Mini project entitled **“AUTOMATED EMERGING CYBER THREAT IDENTIFICATION”** has been submitted by **RAMASWAMI SHERUKULA MOUNIKA (21RG1A05B6), NIMMAKANTI DIVYA (21RG1A05A6), MADDIKUNTA AKHILA (21RG1A0596), THATIPAMULA POOJITHA (21RG1A05C5)** in partial fulfillment of the requirements for the award of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE & ENGINEERING**. This record of bonafide work carried out by them under my guidance and supervision. ***The result embodied in this mini project report has not been submitted to any other University or Institute for the award of any degree.***

#### Mr. Prasanth Kumar Kunda Mrs. K. SHEETAL

Assistant professor Head of the Department

Project Guide

***External Examiner***

*ii*

**ACKNOWLEDGEMENT**

The Mini Project work carried out by our team in the Department of Computer Science and Engineering, Malla Reddy College of Engineering for Women, Hyderabad. ***This work is original and has not been submitted in part or full for any degree or diploma of any other university.***

We wish to acknowledge our sincere thanks to our project guide **Mr. Prasanth Kumar Kunda,** Assistant Professor, Computer Science & Engineering for formulation of the problem, analysis, guidance and her continuous supervision during the course of work.

We acknowledge our sincere thanks to **Dr. Kanaka Durga Returi**, Principal & Professor of Department of Computer Science and Engineering and **Mrs. Sheetal Kulkarni,** Head of the Department, MRCEW and all our faculties ofCSE Departmentfor their kind cooperation in making this Mini Project work a success.

We extend our gratitude to **Sri. Ch. Malla Reddy**, Founder Chairman and **Sri. Ch. Mahender Reddy,** Secretary, **Dr. Vaka Murali Mohan,** Director for their kind cooperation in providing the infrastructure for completion of our Mini Project.

We acknowledge our special thanks to the entire teaching faculty and non-teaching staff members of the Computer Science & Engineering Department for their support in making this project work a success.

**RAMASWAMI SHERUKULA MOUNIKA HT.NO. 21RG1A05B6**

**NIMMAKANTI DIVYA HT.NO. 21RG1A05A6**

**MADDIKUNTA AKHILA HT.NO. 21RG1A0596**

**THATIPAMULA POOJITHA HT.NO. 21RG1A05C5**

iii

# 

# INDEX

|  |  |
| --- | --- |
| **Chapter ABSTRACT**  **LIST OF FIGURES**  **1. SYSTEM ANALYSIS** | **Page No.**  **vii viii**  **1** |
| **1.1 Existing System** | 1 |
| 1.1.1 Disadvantages of Existing System | 4 |
| **1.2 Proposed System** | 4 |
| 1.2.1 Advantages of Proposed System | 5 |
| **1.3 Introduction** | 5 |
| **2. LITERATURE SURVEY** | **7** |
| **3. SYSTEM DESIGN** | **10** |
| 3.1 System modules | 10 |
| 3.1.1 Service provider | 10 |
| 3.1.2 View and Authorized user | 10 |
| 3.1.3 Remote user | 10 |
| 3.2 System Architecture  33.. | 11 |
| 3.3 System Requirements | 12 |
| 3.3.1 Hardware Requirements | 12 |
| 3.3.2 Software Requirements | 12 |
| **3.4 UML Diagrams** | **12** |
| 3.4.1Use Case Diagram | 12 |
| 3.4.2 Class Diagram | 14 |
| 3.4.3 Sequence Diagram | 15 |
| 3.4.4 Flow Chart Diagram | 16 |
| 3.4.5 Date Flow Diagram | 18 |
| **4. INPUT & OUTPUT DESIGN** | **19** |
| 4.1 Input Design    Ivi iiV | 19 |
| 4.2 Output Design | 19 |
| **5. SOFTWARE ENVIRONMENT FOR PYTHON** | **21** |
| 5.1 Python | 21 |
| 5.1.1 Introduction | 21 |
| 5.1.2 History of python | 21 |
| 5.1.3 Python Features | 21 |
| 5.2 Operators | 23 |
| 5.2.1 Arithmetic Operators | 23 |
| 5.2.2 Assignment Operators | 24 |
| 5.2.3 Identity Operators | 26 |
| 5.2.4 Comparison Operators | 26 |
| 5.2.5 Logical Operators | 27 |
| 5.2.6 Membership Operators | 28 |
| 5.3 List | 30 |
| 5.4 Tuples | 32 |
| 5.5 Dictionary | 35 |
| 5.6 Functions | 38 |
| 5.7 Packages in python | 42 |
| 5.8 Exception and its examples | 55 |
| **6. SYSTEM STUDY** | ***60*** |
| 6.1 Economic Feasibility | 60 |
| 6.2 Technical Feasibility | 60 |
| 6.3 Social Feasibility | 60 |
| **7. SYSTEM TESTING** | **62** |
| 7.1 Types of Tests | 62 |
| 7.1.1 Unit testing | 63 |
| 7.1.2 Integration testing | 64 |
| 7.1.3 Acceptance testing | 64 |
| 7.1.4 System testing | 65 |
| 7.1.5 Other testing methodologies | 67 |
| **8. RESULTS** | **71** |
| **9. CONCLUSION** | **82** |
| **10. BIBLIOGRAPHY** | **84** |

# ABSTRACT

The time window between the disclosure of a new cyber vulnerability and its use by cybercriminals has been getting smaller and smaller over time. Recent episodes, such as Log4j vulnerability, exemplifies this well. Within hours after the exploit being released, attackers started scanning the internet looking for vulnerable hosts to deploy threats like cryptocurrency miners and ransomware on vulnerable systems. Thus, it becomes imperative for the cybersecurity defense strategy to detect threats and their capabilities as early as possible to maximize the success of prevention actions. Although crucial, discovering new threats is a challenging activity for security analysts due to the immense volume of data and information sources to be analyzed for signs that a threat is emerging. In this sense, we present a framework for automatic identification and profiling of emerging threats using Twitter messages as a source of events and MITRE ATT&CK as a source of knowledge for threat characterization. The framework comprises three main parts: identification of cyber threats and their names; profiling the identified threat in terms of its intentions or goals by employing two machine learning layers to filter and classify tweets; and alarm generation based on the threat’s risk. The main contribution of our work is the approach to characterize or profile the identified threats in terms of their intentions or goals, providing additional context on the threat and avenues for mitigation. In our experiments, the profiling stage reached an F1 score of 77% in correctly profiling discovered threats.

vii

## LIST OF FIGURES

**Fig. No. Fig. Name Page.no**

|  |  |  |
| --- | --- | --- |
| 3.1 | System Architecture | 11 |
| 3.2 | Use Case Diagram | 13 |
| 3.3 | Class Diagram | 14 |
| 3.4 | Sequence Diagram | 15 |
| 3.5 | Flow Chart Diagram | 16 |
| 3.6 | Flow Chart Diagram | 17 |
| 3.7 | Dataflow Diagram | 18 |

viii

# CHAPTER-1: SYSTEM ANALYSIS

* 1. **EXISTING SYSTEM**
* Cybersecurity is becoming an ever-increasing concern for most organizations and much research has been developed in this field over the last few years. Inside these organizations, the Security Operations Center (SOC) is the central nervous system that provides the necessary security against cyber threats. However, to be effective, the SOC requires timely and relevant threat intelligence to accurately and properly monitor, maintain, and secure an IT infrastructure. This leads security analysts to strive for threat awareness by collecting and reading various information feeds. However, if done manually, this results in a tedious and extensive task that may result in little knowledge being obtained given the large amounts of irrelevant information. Research has shown that Open Source Intelligence (OSINT) provides useful information to identify emerging cyber threats.
* OSINT is the collection, analysis, and use of data from openly available sources for intelligence purposes [21]. Examples of sources for OSINT are public blogs, dark and deep websites, forums, and social media. In such platforms, any person or entity on the Internet can publish, in real-time, information in natural language related to cyber security, including incidents, new threats, and vulnerabilities. Among the OSINT sources for cyber threat intelligence, we can highlight the social media Twitter as one of the most representative [22]. Cyber security experts, system administrators, and hackers constantly use Twitter to discuss technical details about cyber-attacks and share their experiences [4].
* Utilization of OSINT to automatically identify cyber threats via social media, forums and other openly available sources using text analytics was proposed in different researches [1], [23], [7], [24], [25], [26], [13], [27] and [28]. However, most proposals focus on identifying important events related to cyber threats or vulnerabilities but do not propose identifying and profiling cyber threats.
* Amongst research, [13] proposes an early cyber threat warning system that mines online chatter from cyber actors on social media, security blogs, and dark web forums to identify words that signal potential cyber-attacks. The framework is comprised by two main components: text mining and warning generation. The text mining phase consists on pre-processing the input data to identify potential threat names by discarding ‘‘known’’ terms and selecting repeating ‘‘unknown’’ among different sources as they potentially can be the name of a new or discovered cyber threat. The second component, warning generation, is responsible for issuing alarms for unknown terms that meet some requirements, like appearing twice in a given period of time. The approach presented in this research uses keyword filtering as the only strategy to identify cyber threat names, which may result in false positives as unknown words may appear in tweets or other content not necessarily related to cyber security. Additionally, this research does not profile the identified cyber threat.
* In [26] an identification and classification approach of cyber threat indicators in the Twitter stream is presented. The research proposes a data-driven approach for modeling and classification of tweets using a cascaded Convolutional Neural Network (CNN) architecture to both classify tweets as related or not to cyber security and classify the cyber-related tweets into a fixed listed of cyber threats. The proposed solution includes a pre-processing phase that uses IBM’s Watson Natural Language API to identify tweets related to cyber security according to Watson classification results. Additionally, in the pre-processing phase, there is a pre-labeling step performed by simple string matching on the pure tweet text. The threat types considered were: ‘‘vulnerability’’, ‘‘DDoS’’, ‘‘ransomware’’, ‘‘botnet’’, ‘‘data leak’’, ‘‘zero-day’’ and ‘‘general’’. Further, the proposed approach uses CNN models trained to classify each tweet as relevant or irrelevant to cyber security. The relevant tweets are passed to a second CNN layer to be classified as one of the 8 different threat types mentioned above. There are important differences of our proposal compared to this one.
* First, the proposed approach does not name the identified threat. Naming the threat is an important step to cyber threat intelligence as it may allow analysts to identify and mitigate campaigns based on the historic modus operandi employed by a given threat or group.
* Second, the proposed approach relies on an external component to classify tweets as related or not to cyber security as opposed to our approach that proposes a component to classify tweets using machine learning trained with the evolving knowledge from MITRE ATT&CK. Third, instead of using a keyword match to pre-filter threats and a fixed list of threat types, we present an approach to profile the identified cyber threat to spot in which phase of phases of the cyber kill chain the given threat operates in. This is important for a cyber threat analyst as he or she may employ the necessary mitigation steps depending on the threat profile.
* In [1], a framework for automatically gathering cyber threat intelligence from Twitter is presented. The framework utilizes a novelty detection model to classify the tweets as relevant or irrelevant to Cyber threat intelligence. The novelty classifier learns the features of cyber threat intelligence from the threat descriptions in the Common Vulnerabilities and Exposures (CVE) database 5 and classifies a new unseen tweet as normal or abnormal in relation to Cyber threat intelligence. The normal tweets are considered as Cyber threat relevant while the abnormal tweets are considered as Cyber threat-irrelevant. The paper evaluates the framework on a data set created from the tweets collected over a period of twelve months in 2018 from 50 influential Cyber security-related accounts. During the evaluation, the framework achieved the highest performance of 0.643 measured by the F1-score metric for classifying Cyber threat tweets. According to the authors, the proposed approach outperformed several baselines including binary classification models. Also, was analyzed the correctly classified cyber threat tweets and discovered that 81 of them do not contain a CVE identifier. The authors have also found that 34 out of the 81 tweets can be associated with a CVE identifier included in the top 10 most similar CVE descriptions of each tweet. Despite presenting a proposal to distinguish between relevant and irrelevant tweets, the proposal does not address the identification of threats and their intentions. Those are important requirements for Cyber Threat Intelligence in formulating defense strategies against emerging threats.
* The tool proposed in [23] collects tweets from a selected subset of accounts using the Twitter streaming API, and then, by using keyword-based filtering, it discards tweets unrelated to the monitored infrastructure assets. To classify and extract information from tweets the paper uses a sequence of two deep neural networks. The first is a binary classifier based on a Convolutional Neural Network (CNN) architecture used for Natural Language Processing (NLP) [29]. It receives tweets that may be referencing an asset from the monitored infrastructure and labels them as either relevant when the tweets contain security-related information, or irrelevant otherwise.
* Relevant tweets are processed for information extraction by a Named Entity Recognition (NER) model, implemented as a Bidirectional Long Short-Term Memory (BiLSTM) neural network [30]. This network labels each word in a tweet with one of six entities used to locate relevant information. Furthermore, the authors chose to use the application of deep learning techniques because of its advantages in the NLP domain [31]. Thus, they propose an end-to-end threat intelligence tool that relies on neural networks with no feature engineering.

**1.1.1 Disadvantages**

* An existing system never implemented Multi-Class machine learning (ML) algorithms - the next steps in the pipeline.
* An existing system didn’t implement the following method PROCESS IDENTIFIED AND CLASSIFIED THREATS.
  1. **PROPOSED SYSTEM**

The overall goal of this work is to propose an approach to automatically identify and profile emerging cyber threats based on OSINT (Open Source Intelligence) in order to generate timely alerts to cyber security engineers. To achieve this goal, we propose a solution whose macro steps are listed below.

* Continuously monitoring and collecting posts from prominent people and companies on Twitter to mine unknown terms related to cyber threats and malicious campaigns;
* Using Natural Language Processing (NLP) and Machine Learning (ML) to identify those terms most likely to be threat names and discard those least likely;
* Leveraging MITRE ATT&CK techniques’ procedures examples to identify most likely tactic employed by the discovered threat;
* Generating timely alerts for new or developing threats along with its characterization or goals associated with a risk rate based on how fast the threat is evolving since its identification.
  + 1. **Advantages of Proposed System**

To conduct a cyber-attack, malicious actors typically have to

* + Identify vulnerabilities,
  + acquire the necessary tools and tradecraft to successfully exploit them,
  + choose a target and recruit participants,
  + Create or purchase the infrastructure needed, and
  + Plan and execute the attack. Other actors— system administrators, security analysts, and even victims— may discuss vulnerabilities or coordinate a response to attacks
  1. **INTRODUCTION**
* Recently there has been an increasing reliance on the Internet for business, government, and social interactions as a result of a trend of hyper-connectivity and hyper-mobility. While the Internet has become an indispensable infrastructure for businesses, governments, and societies, there is also an increased risk of cyber attacks with different motivations and intentions. Preventing organizations from cyber exploits needs timely intelligence about cyber vulnerabilities and attacks, referred to as threats [1].
* Threat intelligence is defined as ‘‘evidence-based knowledge, including context, mechanisms, indicators, implications, and actionable advice, about an existing or emerging menace or hazard to assets that can be used to inform decisions regarding the subject’s response to that menace or hazard’’ [2]. Threat intelligence in cyber security domain, or cyber threat intelligence, provides timely and relevant information, such as signatures of the attacks, that can help reduce the uncertainty in identifying potential security vulnerabilities and attacks.
* Cyber threat intelligence can generally be extracted from informal or formal sources, which officially release threat information in structured data format. Structured threat intelligence adheres to a well-defined data model, with a common format and structure. Structured cyber threat intelligence, therefore, can be easily parsed by security tools to analyze and respond to security threats accordingly. Examples of formal sources of cyber threat intelligence include the Common Vulnerabilities and Exposures (CVE) database1 and the National Vulnerability Database (NVD).2
* Cyber threat intelligence is also available on informal sources, such as public blogs, dark webs, forums, and social media platforms. Informal sources allow any person or entity on the Internet to publish, in real-time, the threat information in natural language, or unstructured data format. The unstructured and publicly available threat intelligence is also called Open Source Intelligence (OSINT) [3]. Cyber security-related OSINT are early warning sources for cyber security events such as security vulnerability exploits [4].

To conduct a cyber-attack, malicious actors typically have to

* Identify vulnerabilities,
* Acquire the necessary tools and tradecraft to successfully exploit them,
* Choose a target and recruit participants,
* Create or purchase the infrastructure needed, and
* Plan and execute the attack.

Other actors— system administrators, security analysts, and even victims— may discuss vulnerabilities or coordinate a response to attacks [5]. These activities are often conducted online through social media, (open and dark) Web forums, and professional blogs, leaving digital traces behind. Collectively, these digital traces provide valuable insights into evolving cyber threats and can signal a pending or developing attack well before the malicious activity is noted on a target system. For example, exploits are discussed on Twitter before they are publicly disclosed [4] and on dark web forums even before they are discussed on social media [6].

# CHAPTER-2: LITERATURE SURVEY

# A literature survey on automated emerging cyber threat identification reviews the latest research on methods and technologies used to detect new cyber threats. It explores the use of AI, machine learning, anomaly detection, and big data analytics in identifying evolving threats. The survey typically highlights key techniques like behavioral analysis, threat intelligence integration, and real-time monitoring, as well as challenges such as data accuracy, false positives, and adaptability to novel attack patterns. This research provides insights into advancements and gaps in automating cyber threat detection.

# Artificial intelligence for cyber security: Literature review and future research directions Abstract: Artificial intelligence (AI) is a powerful technology that helps cyber security teams automate repetitive tasks, accelerate threat detection and response, and improve the accuracy of their actions to strengthen the security posture against various security issues and cyber attacks. This article presents a systematic literature review and a detailed analysis of AI use cases for cyber security provisioning. The review resulted in 2395 studies, of which 236 were identified as primary. This article classifies the identified AI use cases based on a NIST cyber security framework using a thematic analysis approach. This classification framework will provide readers with a comprehensive overview of the potential of AI to improve cyber security in different contexts. The review also identifies future research opportunities in emerging cyber security application areas, advanced AI methods, data representation, and the development of new infrastructures for the successful adoption of AI-based cyber security in today's era of digital transformation and polycrisis.

# NLP-Based Techniques for Cyber Threat Intelligence Abstract: In the digital era, threat actors employ sophisticated techniques for which, often, digital traces in the form of textual data are available. Cyber Threat Intelligence (CTI) is related to all the solutions inherent to data collection, processing, and analysis useful to understand a threat actor’s targets and attack behavior. Currently, CTI is assuming an always more crucial role in identifying and mitigating threats and enabling proactive defense strategies. In this context, NLP, an artificial intelligence branch, has emerged as a powerful tool for enhancing threat intelligence capabilities. This survey paper provides a comprehensive overview of NLP-based techniques applied in the context of threat intelligence. It begins by describing the foundational definitions and principles of CTI as a major tool for safeguarding digital assets. It then undertakes a thorough examination of NLP based techniques for CTI data crawling from Web sources, CTI data analysis, Relation Extraction from cyber security Data, CTI sharing and collaboration, and security threats of CTI. Finally, the challenges and limitations of NLP in threat intelligence are exhaustively examined, including data quality issues and ethical considerations. This survey draws a complete framework and serves as a valuable resource for security professionals and researchers seeking to understand the state-of-the-art NLP-based threat intelligence techniques and their potential impact on cyber security.

# A Systematic Literature Review on Cyber Threat Intelligence for Organizational Cyber security Resilience Abstract: Cyber security is a significant concern for businesses worldwide, as cybercriminals target business data and system resources. Cyber threat intelligence (CTI) enhances organizational cyber security resilience by obtaining, processing, evaluating, and disseminating information about potential risks and opportunities inside the cyber domain. This research investigates how companies can employ CTI to improve their precautionary measures against security breaches. The study follows a systematic review methodology, including selecting primary studies based on specific criteria and quality valuation of the selected papers. As a result, a comprehensive framework is proposed for implementing CTI in organizations. The proposed framework is comprised of a knowledge base, detection models, and visualization dashboards. The detection model layer consists of behavior-based, signature-based, and anomaly based detection. In contrast, the knowledge base layer contains information resources on possible threats, vulnerabilities, and dangers to key assets. The visualization dashboard layer provides an overview of key metrics related to cyber threats, such as an organizational risk meter, the number of attacks detected, types of attacks, and their severity level. This relevant systematic study also provides insight for future studies, such as how organizations can tailor their approach to their needs and resources to facilitate more effective collaboration between stakeholders while navigating legal/regulatory constraints related to information sharing.

# Explainable artificial intelligence for cybersecurity: a literature survey Abstract: With the extensive application of deep learning (DL) algorithms in recent years, e.g., for detecting Android malware or vulnerable source code, artificial intelligence (AI) and machine learning (ML) are increasingly becoming essential in the development of cybersecurity solutions. However, sharing the same fundamental limitation with other DL application domains, such as computer vision (CV) and natural language processing (NLP), AI-based cybersecurity solutions are incapable of justifying the results (ranging from detection and prediction to reasoning and decision-making) and making them understandable to humans. Consequently, explainable AI (XAI) has emerged as a paramount topic addressing the related challenges of making AI models explainable or interpretable to human users. It is particularly relevant in cyber security domain, in that XAI may allow security operators, who are overwhelmed with tens of thousands of security alerts per day (most of which are false positives), to better assess the potential threats and reduce alert fatigue. We conduct an extensive literature review on the intersection between XAI and cyber security. Particularly, we investigate the existing literature from two perspectives: the applications of XAI to cyber security (e.g., intrusion detection, malware classification), and the security of XAI (e.g., attacks on XAI pipelines, potential countermeasures). We characterize the security of XAI with several security properties that have been discussed in the literature. We also formulate open questions that are either unanswered or insufficiently addressed in the literature, and discuss future directions of research.

# 

# CHAPTER-3: SYSTEM DESIGN

**3.1 System Modules**

**3.1.1 Service provider:**

In this module, the Service Provider has to login by using valid user name and password. After login successful he can do some operations, such as Browse Datasets and Train & Test Data Sets, View Trained and Tested Accuracy in Bar Chart, View Trained and Tested Accuracy Results, View Prediction of Cyber Threat Identification Type, View Cyber Threat Identification Type Ratio, Download Predicted Data Sets, View Cyber Threat Identification Type Ratio Results, View All Remote Users.

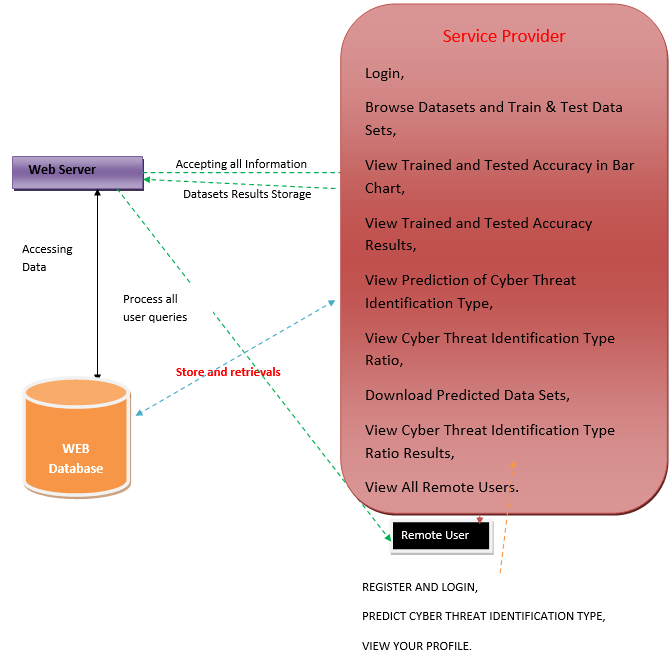
**3.1.2 View and Authorized user**

In this module, the admin can view the list of users who all registered. In this, the admin can view the user’s details such as, user name, email, address and admin authorizes the users.

**3.1.3 Remote user**

In this module, there are n numbers of users are present. User should register before doing any operations. Once user registers, their details will be stored to the database. After registration successful, he has to login by using authorized user name and password. Once Login is successful user will do some operations like REGISTER AND LOGIN, PREDICT CYBER THREAT IDENTIFICATION TYPE, VIEW YOUR PROFILE.

**3.2 System Architecture**



**Fig.3.1: System Architecture**

## System Requirements

**3.3.1** **Hardware System Configuration: -**

➢ Processor - Pentium –IV

➢ RAM - 4 GB (min)

➢ Hard Disk - 20 GB

➢ Key Board - Standard Windows Keyboard

➢ Mouse - Two or Three Button Mouse

➢ Monitor - SVGA

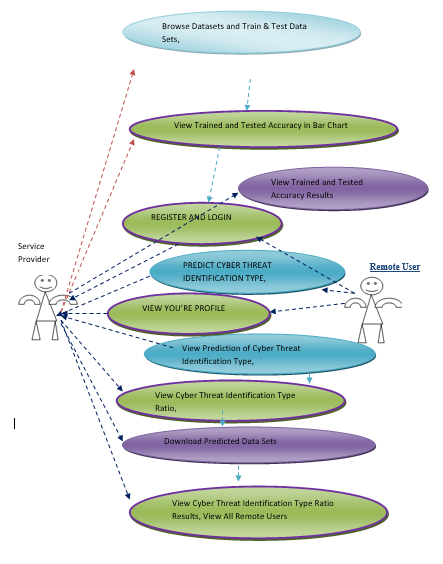
**3.3.2: Software Requirements: -**

* Operating system  **:** Windows 7 Ultimate.
* Coding Language **:** Python.
* Front-End **:** Python.
* Back-End **:** Django-ORM
* Designing **:** Html, CSS, JavaScript.
* DataBase **:** MySQL (WAMP Server).

**3.4 UML Diagrams**

* + 1. **Use Case Diagram**

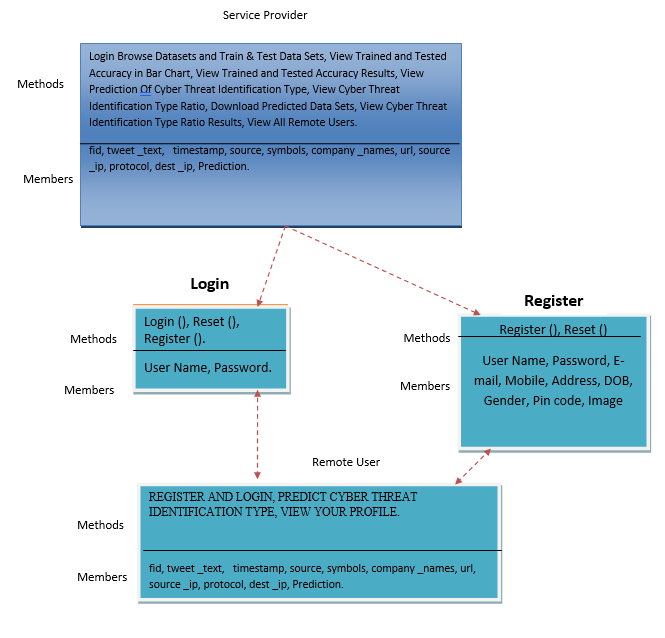
In the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those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



**Fig. 3.2: Use Case Diagrams**

* + 1. **Class Diagram**

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

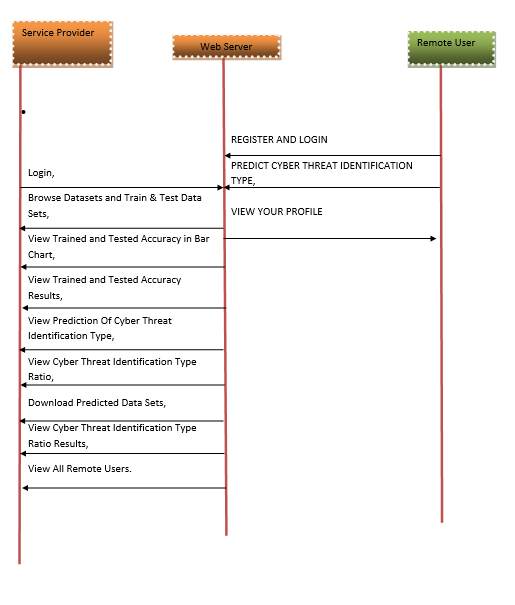


## Fig. 3.3: Class Diagram

* + 1. **Sequence Diagram**

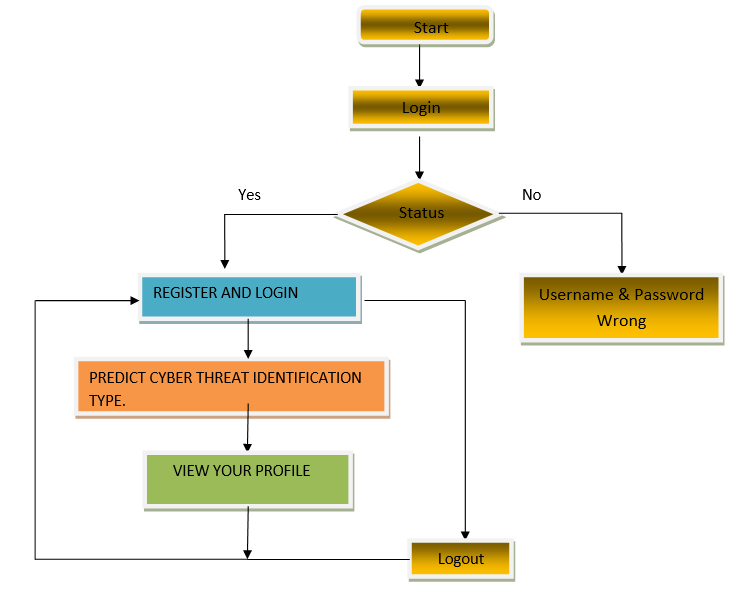
A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart.

Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

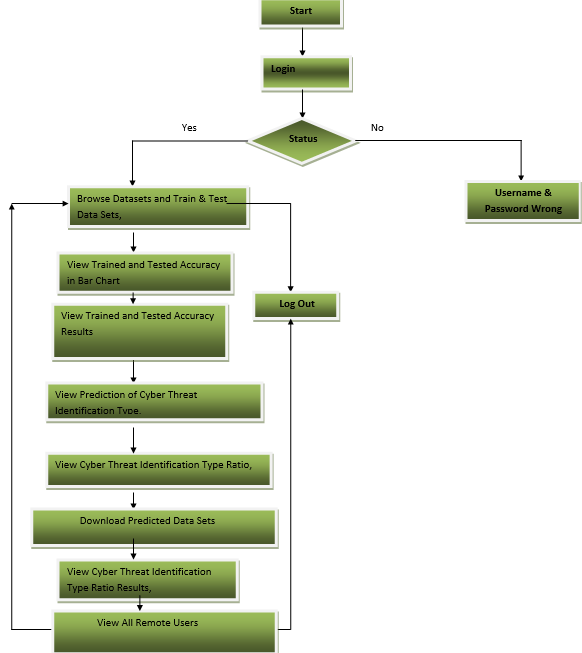


## Fig. 3.4: Sequence Diagram

* + 1. **Flow Chart Diagram**

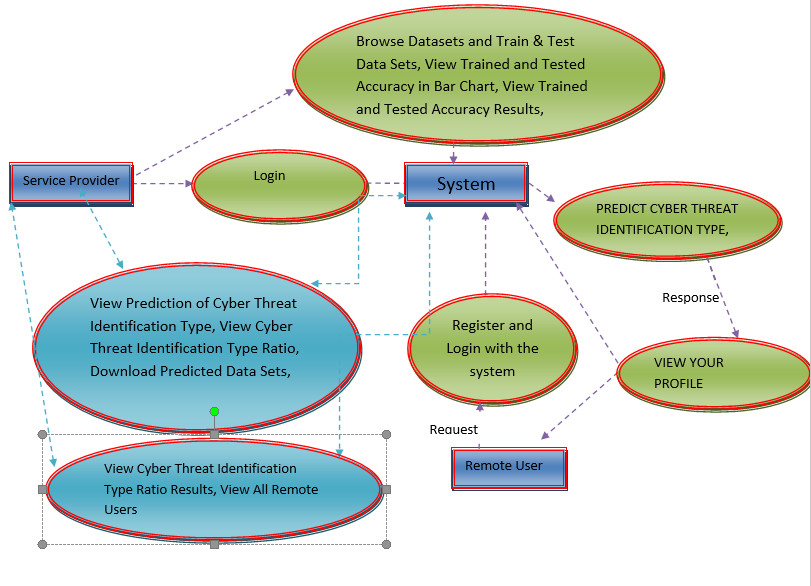


**Fig.3.5: Flow Chart Diagram of Remote user**



**Fig.3.6: Flow Chart Diagram of service provider**

* + 1. **Data Flow Diagram**



**Fig.3.7: Data Flow Diagram**

**CHAPTER-4: INPUT AND OUTPUT DESIGN**

**4.1 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.

**4.2 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.

# CHAPTER-5: SOFTWARE ENVIRONMENT

**5.1 PYTHON**

**5.1.1 Introduction**

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.

## 5.1.2 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.

## 5.1.3 Python Features

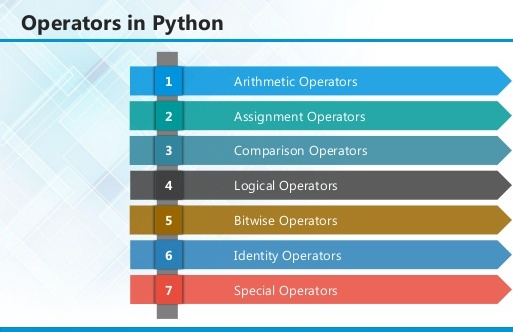
Python's features include:

* **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.
* **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
* **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.

**5.2 Operators**



**5.2.1 ARITHMETIC OPERATORS**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + Addition | Adds values on either side of the operator. | a + b = 30 |
| - Subtraction | Subtracts right hand operand from left hand operand. | a – b = -10 |
| \* Multiplication | Multiplies values on either side of the operator | a \* b = 200 |
| / Division | Divides left hand operand by right hand operand | b / a = 2 |
| % Modulus | Divides left hand operand by right hand operand and returns remainder | b % a = 0 |
| \*\* Exponent | Performs exponential (power) calculation on operators | a\*\*b =10 to the power 20 |
| // | Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity): | 9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0 |

**5.2.2 ASSIGNMENT OPERATORS**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| = | Assigns values from right side operands to left side operand | c = a + b assigns value of a + b into c |
| += Add AND | It adds right operand to the left operand and assign the result to left operand | c += a is equivalent to c = c + a |
| -= Subtract AND | It subtracts right operand from the left operand and assign the result to left operand | c -= a is equivalent to c = c - a |
| \*= Multiply AND | It multiplies right operand with the left operand and assign the result to left operand | c \*= a is equivalent to c = c \* a |
| /= Divide AND | It divides left operand with the right operand and assign the result to left operand | c /= a is equivalent to c = c / ac /= a is equivalent to c = c / a |

|  |  |  |
| --- | --- | --- |
| %= Modulus AND | It takes modulus using two operands and assign the result to left operand | c %= a is equivalent to c = c % a |
| \*\*= Exponent AND | Performs exponential (power) calculation on operators and assign value to the left operand | c \*\*= a is equivalent to c = c \*\* a |
| //= Floor Division | It performs floor division on operators and assign value to the left operand | c //= a is equivalent to c = c // a |

**5.2.3 IDENTITY OPERATORS**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| is | Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. | x is y, here **is** results in 1 if id(x) equals id(y). |
| is not | Evaluates to false if the variables on either side of the operator point to the same object and true otherwise. | x is not y, here **is not** results in 1 if id(x) is not equal to id(y |

**5.2.4 COMPARISON OPERATORS**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| & Binary AND | Operator copies a bit to the result if it exists in both operands | (a & b) (means 0000 1100) |
| | Binary OR | It copies a bit if it exists in either operand. | (a | b) = 61 (means 0011 1101) |
| ^ Binary XOR | It copies the bit if it is set in one operand but not both. | (a ^ b) = 49 (means 0011 0001) |
| ~ Binary Ones Complement | It is unary and has the effect of 'flipping' bits. | (~a ) = -61 (means 1100 0011 in 2's complement form due to a signed binary number. |
| << Binary Left Shift | The left operands value is moved left by the number of bits specified by the right operand. | a << 2 = 240 (means 1111 0000) |
| >> Binary Right Shift | The left operands value is moved right by the number of bits specified by the right operand. | a >> 2 = 15 (means 0000 1111) |

**5.2.5 LOGICAL OPERATORS**

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| and Logical AND | If both the operands are true then condition becomes true. | (a and b) is true. |
| or Logical OR | If any of the two operands are non-zero then condition becomes true. | (a or b) is true. |
| not Logical NOT | Used to reverse the logical state of its operand. | Not(a and b) is false. |

## 5.2.6 MEMBERSHIP OPERATORS

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| in | Evaluates to true if it finds a variable in the specified sequence and false otherwise. | x in y, here in results in a 1 if x is a member of sequence y. |
| not in | Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. | x not in y, here not in results in a 1 if x is not a member of sequence y. |

## Python Operators Precedence

|  |  |
| --- | --- |
| **Operator** | **Description** |
| \*\* | Exponentiation (raise to the power) |
| ~ + - | Complement, unary plus and minus (method names for the last two are +@ and -@) |
| \* / % // | Multiply, divide, modulo and floor division |
| + - | Addition and subtraction |
| >> << | Right and left bitwise shift |
| & | Bitwise 'AND' |  |
| ^ | | Bitwise exclusive `OR' and regular `OR' |  |
| <= < > >= | Comparison operators |  |
| <> == != | Equality operators |  |
| = %= /= //= -= += \*= \*\*= | Assignment operators |  |
| is is not | Identity operators |  |
| in not in | Membership operators |  |
| not or and | Logical operators |  |

**5.3 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)](https://www.tutorialspoint.com/python/list_cmp.htm)  Compares elements of both lists. |
| 2 | [len(list)](https://www.tutorialspoint.com/python/list_len.htm)  Gives the total length of the list. |
| 3 | [max(list)](https://www.tutorialspoint.com/python/list_max.htm)  Returns item from the list with max value. |
| 4 | [min(list)](https://www.tutorialspoint.com/python/list_min.htm)  Returns item from the list with min value. |
| 5 | [list(seq)](https://www.tutorialspoint.com/python/list_list.htm)  Converts a tuple into list. |

Python includes following list methods

|  |  |
| --- | --- |
| **SN** | **Methods with Description** |
| 1 | [list.append(obj)](https://www.tutorialspoint.com/python/list_append.htm)  Appends object obj to list |
| 2 | [list.count(obj)](https://www.tutorialspoint.com/python/list_count.htm)  Returns count of how many times obj occurs in list |
| 3 | [list. extend(seq)](https://www.tutorialspoint.com/python/list_extend.htm)  Appends the contents of seq to list |
| 4 | [list.index(obj)](https://www.tutorialspoint.com/python/list_index.htm)  Returns the lowest index in list that obj appears |
| 5 | [list.insert(index, obj)](https://www.tutorialspoint.com/python/list_insert.htm)  Inserts object obj into list at offset index |
| 6 | [list.pop(obj=list[-1])](https://www.tutorialspoint.com/python/list_pop.htm)  Removes and returns last object or obj from list |
| 7 | [list.remove(obj)](https://www.tutorialspoint.com/python/list_remove.htm)  Removes object obj from list |
| 8 | [list.reverse()](https://www.tutorialspoint.com/python/list_reverse.htm)  Reverses objects of list in place |
| 9 | [list.sort([func])](https://www.tutorialspoint.com/python/list_sort.htm)  Sorts objects of list, use compare function if given |

**5.4 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** | **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 Tuple Functions

|  |  |
| --- | --- |
| **SN** | **Function with Description** |
| 1 | **[cmp(tuple1, tuple2)](https://www.tutorialspoint.com/python/tuple_cmp.htm)**:Compares elements of both tuples. |
| 2 | **[len(tuple)](https://www.tutorialspoint.com/python/tuple_len.htm)**:Gives the total length of the tuple. |
| 3 | **[max(tuple)](https://www.tutorialspoint.com/python/tuple_max.htm)**:Returns item from the tuple with max value. |
| 4 | **[min(tuple)](https://www.tutorialspoint.com/python/tuple_min.htm)**:Returns item from the tuple with min value. |
| 5 | **[tuple(seq)](https://www.tutorialspoint.com/python/tuple_tuple.htm)**:Converts a list into tuple. |

**5.5 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)](https://www.tutorialspoint.com/python/dictionary_cmp.htm)  Compares elements of both dict. |
| 2 | [len(dict)](https://www.tutorialspoint.com/python/dictionary_len.htm)  Gives the total length of the dictionary. This would be equal to the number of items in the dictionary. |
| 3 | [str(dict)](https://www.tutorialspoint.com/python/dictionary_str.htm)  Produces a printable string representation of a dictionary |
| 4 | [type(variable)](https://www.tutorialspoint.com/python/dictionary_type.htm)  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()](https://www.tutorialspoint.com/python/dictionary_clear.htm)**:Removes all elements of dictionary *dict* |
| 2 | **[dict. Copy()](https://www.tutorialspoint.com/python/dictionary_copy.htm)**:Returns a shallow copy of dictionary *dict* |
| 3 | **[dict.fromkeys()](https://www.tutorialspoint.com/python/dictionary_fromkeys.htm)**:Create a new dictionary with keys from seq and values *set* to *value*. |
| 4 | **[dict.get(key, default=None)](https://www.tutorialspoint.com/python/dictionary_get.htm)**:For *key* key, returns value or default if key not in dictionary |
| 5 | **[dict.has\_key(key)](https://www.tutorialspoint.com/python/dictionary_has_key.htm)**:Returns *true* if key in dictionary *dict*, *false* otherwise |
| 6 | **[dict.items()](https://www.tutorialspoint.com/python/dictionary_items.htm)**:Returns a list of *dict*'s (key, value) tuple pairs |
| 7 | **[dict.keys()](https://www.tutorialspoint.com/python/dictionary_keys.htm)**:Returns list of dictionary dict's keys |
| 8 | **[dict.setdefault(key, default=None)](https://www.tutorialspoint.com/python/dictionary_setdefault.htm)**:Similar to get(), but will set dict[key]=default if *key* is not already in dict |
| 9 | **[dict.update(dict2)](https://www.tutorialspoint.com/python/dictionary_update.htm)**:Adds dictionary *dict2*'s key-values pairs to *dict* |
| 10 | **[dict.values()](https://www.tutorialspoint.com/python/dictionary_values.htm)**:Returns list of dictionary *dict*'s values |

**5.6 Functions**

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.

## 5.7 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

## File Positions

The *tell()* method tells you the current position within the file; in other words, the next read or write will occur at that many bytes from the beginning of the file.

32

The *seek(offset[, from])* method changes the current file position. The *offset* argument indicates the number of bytes to be moved. The *from* argument specifies the reference position from where the bytes are to be moved.

If *from* is set to 0, it means use the beginning of the file as the reference position and 1 means use the current position as the reference position and if it is set to 2 then the end of the file would be taken as the reference position.

### Example

Let us 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

# Check current position

position = fo.tell();

print "Current file position : ", position

# Reposition pointer at the beginning once again

position = fo.seek(0, 0);

str = fo.read(10);

print "Again read String is : ", str

# Close opend file

fo.close()

This produces the following result −

Read String is : Python is

Current file position : 10

Again read String is : Python is

## Renaming and Deleting Files

Python **os** module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then you can call any related functions.

## The rename() Method

The *rename()* method takes two arguments, the current filename and the new filename.

### Syntax

os.rename(current\_file\_name, new\_file\_name)

### Example

Following is the example to rename an existing file *test1.txt*:

import os

# Rename a file from test1.txt to test2.txt

os.rename( "test1.txt", "test2.txt" )

## The *remove()* Method

You can use the *remove()* method to delete files by supplying the name of the file to be deleted as the argument.

### Syntax

os.remove(file\_name)

### Example

Following is the example to delete an existing file *test2.txt* −

#!/usr/bin/python

import os

# Delete file test2.txt

os.remove("text2.txt")

## Directories in Python

All files are contained within various directories, and Python has no problem handling these too. The **os** module has several methods that help you create, remove, and change directories.

## The *mkdir()* Method

You can use the *mkdir()* method of the **os** module to create directories in the current directory. You need to supply an argument to this method which contains the name of the directory to be created.

### Syntax

os.mkdir("newdir")

### Example

Following is the example to create a directory *test* in the current directory −

#!/usr/bin/python

import os

# 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

# This would give location of the current directory

os.getcwd()

## The *rmdir()* Method

The *rmdir()* method deletes the directory, which is passed as an argument in the method.

Before removing a directory, all the contents in it should be removed.

### Syntax:

os.rmdir('dirname')

### Example

Following is the example to remove "/tmp/test" directory. It is required to give fully qualified name of the directory, otherwise it would search for that directory in the current directory.

import os

# This would remove "/tmp/test" directory.

os.rmdir( "/tmp/test" )

## File & Directory Related Methods

There are three important sources, which provide a wide range of utility methods to handle and manipulate files & directories on Windows and Unix operating systems. They are as follows −

* [File Object Methods](https://www.tutorialspoint.com/python/file_methods.htm): The *file* object provides functions to manipulate files.
* [OS Object Methods](https://www.tutorialspoint.com/python/os_file_methods.htm): This provides methods to process files as well as directories.

Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them −

* **Exception Handling:** This would be covered in this tutorial. Here is a list standard Exceptions available in Python: [Standard Exceptions](https://www.tutorialspoint.com/python/standard_exceptions.htm).
* **Assertions:** This would be covered in [Assertions in Python](https://www.tutorialspoint.com/python/assertions_in_python.htm)

**5.8 List of Standard Exceptions −**

|  |  |
| --- | --- |
| **EXCEPTION NAME** | **DESCRIPTION** |
| Exception | Base class for all exceptions |
| StopIteration | Raised when the next() method of an iterator does not point to any object. |
| SystemExit | Raised by the sys.exit() function. |
| StandardError | Base class for all built-in exceptions except StopIteration and SystemExit. |
| ArithmeticError | Base class for all errors that occur for numeric calculation. |
| OverflowError | Raised when a calculation exceeds maximum limit for a numeric type. |
| FloatingPointError | Raised when a floating point calculation fails. |
| ZeroDivisionError | Raised when division or modulo by zero takes place for all numeric types. |
| AssertionError | Raised in case of failure of the Assert statement. |
| AttributeError | Raised in case of failure of attribute reference or assignment. |
| EOFError | Raised when there is no input from either the raw\_input() or input() function and the end of file is reached. |
| ImportError | Raised when an import statement fails. |
| KeyboardInterrupt | Raised when the user interrupts program execution, usually by pressing Ctrl+c. |
| LookupError | Base class for all lookup errors. |
| IndexError  KeyError | Raised when an index is not found in a sequence.  Raised when the specified key is not found in the dictionary. |
| NameError | Raised when an identifier is not found in the local or global namespace. |
| UnboundLocalError  EnvironmentError | Raised when trying to access a local variable in a function or method but no value has been assigned to it.  Base class for all exceptions that occur outside the Python environment. |
| IOError  IOError | Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.  Raised for operating system-related errors. |
| SyntaxError  IndentationError | Raised when there is an error in Python syntax.  Raised when indentation is not specified properly. |
| SystemError | Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit. |
| SystemExit | Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit. |
| TypeError | Raised when an operation or function is attempted that is invalid for the specified data type. |
| ValueError | Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified. |
| RuntimeError | Raised when a generated error does not fall into any category. |
| NotImplementedError | Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented. |

## What is Exception?

An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object that represents an error.

When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

## Handling an exception

If you have some *suspicious* code that may raise an exception, you can defend your program by placing the suspicious code in a **try:** block. After the try: block, include an **except:** statement, followed by a block of code which handles the problem as elegantly as possible.

The Python standard for database interfaces is the Python DB-API. Most Python database interfaces adhere to this standard.

You can choose the right database for your application. Python Database API supports a wide range of database servers such as −

* GadFly
* mSQL
* MySQL
* PostgreSQL
* Microsoft SQL Server 2000
* Informix
* Interbase
* Oracle
* Sybase

The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following:

* Importing the API module.
* Acquiring a connection with the database.
* Issuing SQL statements and stored procedures.
* Closing the connection

# CHAPTER-6: 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
  1. **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.

* 1. **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.

* 1. **SOCIAL FEASIBILITY**

The aspect of study is to check the level of acceptance of the system by t 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: 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**.**

* 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.

* + 1. **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.

**7.1.2. 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.

* + 1. **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.

**7.1.4 SYSTEM TESTING**

**TESTING METHODOLOGIES**

The following are the Testing Methodologies:

**Unit Testing.**

**Integration Testing.**

**User Acceptance Testing.**

**Output Testing.**

**Validation Testing.**

**Unit Testing**

Unit testing focuses verification effort on the smallest unit of Software design that is the module. Unit testing exercises specific paths in a module’s control structure to ensure complete coverage and maximum error detection. This test focuses on each module individually, ensuring that it functions properly as a unit. Hence, the naming is Unit Testing.

During this testing, each module is tested individually and the module interfaces are verified for the consistency with design specification. All important processing path are tested for the expected results. All error handling paths are also tested.

**Integration Testing**

Integration testing addresses the issues associated with the dual problems of verification and program construction. After the software has been integrated a set of high order tests are conducted. The main objective in this testing process is to take unit tested modules and builds a program structure that has been dictated by design.

**The following are the types of Integration Testing:**

**1)Top Down Integration**

This method is an incremental approach to the construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main program module. The module subordinates to the main program module are incorporated into the structure in either a depth first or breadth first manner.

In this method, the software is tested from main module and individual stubs are replaced when the test proceeds downwards.

**2. Bottom-up Integration**

This method begins the construction and testing with the modules at the lowest level in the program structure. Since the modules are integrated from the bottom up, processing required for modules subordinate to a given level is always available and the need for stubs is eliminated. The bottom up integration strategy may be implemented with the following steps:

The low-level modules are combined into clusters into clusters that perform a specific Software sub-function.

A driver (i.e.) the control program for testing is written to coordinate test case input and output.

The cluster is tested.

Drivers are removed and clusters are combined moving upward in the program structure

The bottom up approaches tests each module individually and then each module is module is integrated with a main module and tested for functionality.

* + 1. **OTHER TESTING METHODOLOGIES**

**User Acceptance Testing**

User Acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required. The system developed provides a friendly user interface that can easily be understood even by a person who is new to the system.

**Output Testing**

After performing the validation testing, the next step is output testing of the proposed system, since no system could be useful if it does not produce the required output in the specified format. Asking the users about the format required by them tests the outputs generated or displayed by the system under consideration. Hence the output format is considered in 2 ways – one is on screen and another in printed format.

**Validation Checking**

Validation checks are performed on the following fields.

**Text Field:**

The text field can contain only the number of characters lesser than or equal to its size. The text fields are alphanumeric in some tables and alphabetic in other tables. Incorrect entry always flashes and error message.

**Numeric Field:**

The numeric field can contain only numbers from 0 to 9. An entry of any character flashes an error messages. The individual modules are checked for accuracy and what it has to perform. Each module is subjected to test run along with sample data. The individually tested modules are integrated into a single system. Testing involves executing the real data information is used in the program the existence of any program defect is inferred from the output. The testing should be planned so that all the requirements are individually tested.

A successful test is one that gives out the defects for the inappropriate data and produces and output revealing the errors in the system.

**Preparation of Test Data**

Taking various kinds of test data does the above testing. Preparation of test data plays a vital role in the system testing. After preparing the test data the system under study is tested using that test data. While testing the system by using test data errors are again uncovered and corrected by using above testing steps and corrections are also noted for future use.

**Using Live Test Data:**

Live test data are those that are actually extracted from organization files. After a system is partially constructed, programmers or analysts often ask users to key in a set of data from their normal activities. Then, the systems person uses this data as a way to partially test the system. In other instances, programmers or analysts extract a set of live data from the files and have them entered themselves.

It is difficult to obtain live data in sufficient amounts to conduct extensive testing. And, although it is realistic data that will show how the system will perform for the typical processing requirement, assuming that the live data entered are in fact typical, such data generally will not test all combinations or formats that can enter the system. This bias toward typical values then does not provide a true systems test and in fact ignores the cases most likely to cause system failure.

**Using Artificial Test Data:**

Artificial test data are created solely for test purposes, since they can be generated to test all combinations of formats and values. In other words, the artificial data, which can quickly be prepared by a data generating utility program in the information systems department, make possible the testing of all login and control paths through the program.

The most effective test programs use artificial test data generated by persons other than those who wrote the programs. Often, an independent team of testers formulates testing plan, using the systems specifications.

The package “Virtual Private Network” has satisfied all the requirements specified as per software requirement specification and was accepted.

**USER TRAINING**

Whenever a new system is developed, user training is required to educate them about the working of the system so that it can be put to efficient use by those for whom the system has been primarily designed. For this purpose the normal working of the project was demonstrated to the prospective users. Its working is easily understandable and since the expected users are people who have good knowledge of computers, the use of this system is very easy.

**MAINTAINENCE**

This covers a wide range of activities including correcting code and design errors. To reduce the need for maintenance in the long run, we have more accurately defined the user’s requirements during the process of system development. Depending on the requirements, this system has been developed to satisfy the needs to the largest possible extent. With development in technology, it may be possible to add many more features based on the requirements in future. The coding and designing is simple and easy to understand which will make maintenance easier.

**TESTING STRATEGY :**

A strategy for system testing integrates system test cases and design techniques into a well planned series of steps that results in the successful construction of software. The testing strategy must co-operate test planning, test case design, test execution, and the resultant data collection and evaluation .A strategy for software testing must accommodate low-level tests that are necessary to verify that a small source code segment has been correctly implemented as well as high level tests that validate major system functions against user requirements.

Software testing is a critical element of software quality assurance and represents the ultimate review of specification design and coding. Testing represents an interesting anomaly for the software. Thus, a series of testing are performed for the proposed system before the system is ready for user acceptance testing.

**SYSTEM TESTING:**

Software once validated must be combined with other system elements (e.g. Hardware, people, database). System testing verifies that all the elements are proper and that overall system function performance is achieved. It also tests to find discrepancies between the system and its original objective, current specifications and system documentation.

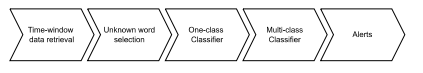
**UNIT TESTING:**

In unit testing different are modules are tested against the specifications produced during the design for the modules. Unit testing is essential for verification of the code produced during the coding phase, and hence the goals to test the internal logic of the modules. Using the detailed design description as a guide, important Conrail paths are tested to uncover errors within the boundary of the modules. This testing is carried out during the programming stage itself. In this type of testing step, each module was found to be working satisfactorily as regards to the expected output from the module.

In Due Course, latest technology advancements will be taken into consideration. As part of technical build-up many components of the networking system will be generic in nature so that future projects can either use or interact with this.The future holds a lot to offer to the development and refinement of this project.

# CHAPTER-8: RESULTS

This chapter presents the results achieved by running our proposed solution, since the data retrieval from the tweets database to the generated alerts. The implementation of the pipeline was automated with scripts developed in Python. To make it easier to follow each step of the pipeline and its results, we are going to use the diagram in Figure, which represents the macro steps of the pipeline, highlighting each step in the following subsections. The full pipeline is described in section III-A.

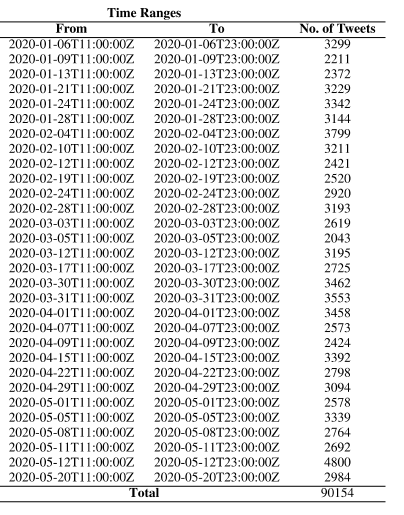


**Proposed solution pipeline macro diagram.**

**A.TIME-WINDOW DATA RETRIEVAL**

This is the first step of our pipeline. It consists in collecting data from the Tweets Database, described in section III-A2, for a given time range. When in production, the time range will be a sliding time window. However, for the purposes of this experiment, we considered 30 aleatory 12 hours intervals of time for weekdays from the first semester of 2020. The objective is to simulate the execution of the system in those different time windows and check the results. The time ranges considered and the total tweets collected for each day are shown in Table 5. The total number of tweets collected for the experiment was 90154.

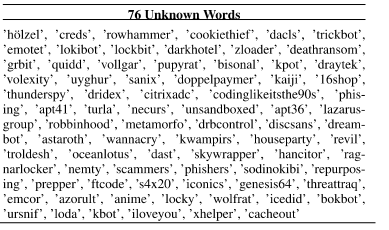
**TABLE 5. Tweets time ranges and a respective number of messages.**



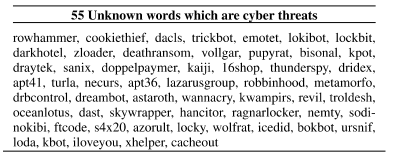
**B. UNKNOWN WORD IDENTIFICATION**

The step of Unknown word selection, described in section III-A4, consists in identifying words mentioned in tweets that are potential names of cyber threats. Each set of tweets resulting from the previous step passed through the stage of the Unknown Word Selection step of the pipeline. In the end, this process resulted in 76 unknown words which appeared in 241 tweets. The unknown words are listed in Table 6

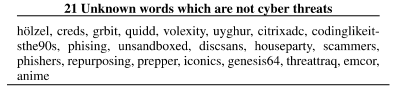
**TABLE 6. List of unknown words identified.**



**TABLE 7. List of unknown words which are cyber threats.**

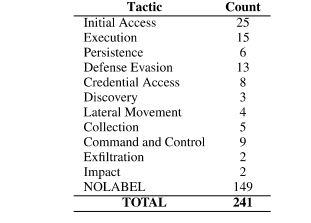


**TABLE 8. List of unknown words which are not cyber threats.**

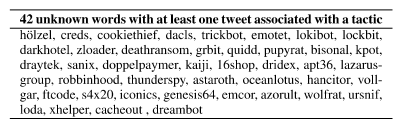


An experienced cyber security specialist verified the resulting list of unknown words to verify which of those are cyber threats and which are not. The verification showed that, from 76 unknown words selected by the ‘Unknown Word Selection’ step, 56 are threats and 20 are not as shown in Tables 7 and 8, respectively. 1) DATA LABELLING An additional procedure was taken in this step of the pipeline to prepare the data to be validated by the next two steps of the pipeline, which consist in classifying data using two machine learning models. The additional step consisted in labeling each of 241 Twitter messages returned by ‘Unknown Word Selection’. The labeling was performed by a cyber security specialist that read carefully each tweet message and associated a label according to the following rules: 1) If the tweet mentions or describes, even partially, one of the 14 MITRE ATT&CK tactics, a label with the corresponding tactic is associated with the Tweet message; 2) If the tweet message does not describe any of MITRE ATT&CK tactics, a label ‘NOLABEL’ was associated. To make the labeling job simpler and more organized, we use a tool called Doccano.23 Doccano is an open-source text annotation tool to create labeled data for sentiment analysis, named entity recognition, text summarization, and so on. In Figure 5, there is an example of a Tweet message being labeled using Doccano tool. The message says that Cookiethief, the discovered threat, exfiltrates browser and Facebook app cookies to a malicious server. Thus the ‘Exfiltration’ tactic was associated with the message. By the end of the labeling process, from 241 tweets, 92 tweets were associated with a MITRE ATT&CK tactic and 149 were not (NOLABEL), as seen in Table 9. From the total of 76 unknown words, 42 are present in the 92 tweets associated with at least one Mitre ATT&CK tactic .

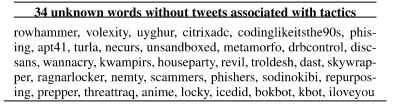
**TABLE 9. Tweets labelling results classified by MITRE ATT&CK Tactic.**

****

**TABLE 10. Unknown words with at least one tweet associated with a tactic.**

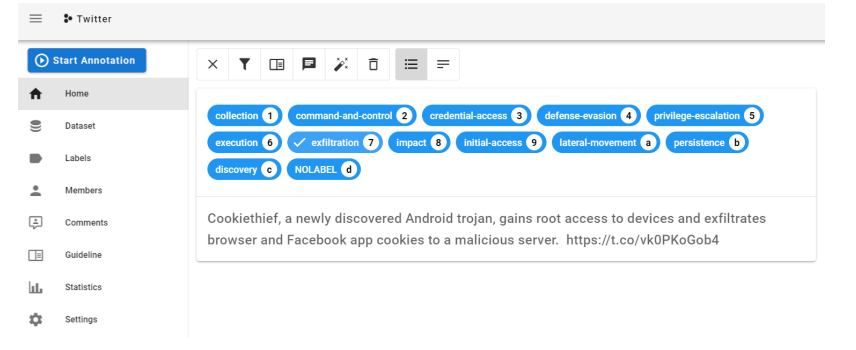
****

**TABLE 11. Unknown words without tweets associated with tactics**

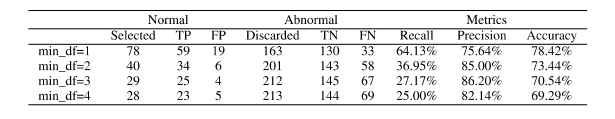
****

**C. ONE-CLASS CLASSIFICATION**

This step of the pipeline consists in using a machine learning model to select tweets whose content is close to the description of malicious actions while discarding the ones that are not. To do this, we implemented a One-class classifier, as detailed in III-A8. The model decides whether a tweet message is normalsimilar or abnormaldifferent from the training set. The normal messages are selected and forwarded to the next step of the pipeline and the abnormal messages are discarded. As a reminder, the training set for the One-class model consists of all the MITRE ATT&CK techniques’ threat procedures examples. The One-class classifier was implemented using OneClassSVM algorithm from SkitLearn and was trained with all the procedures of MITRE ATT&CK, regardless of the tactic or technique to which it belongs, as described in section III-A8. The training data consists of a TF-IDF feature matrix created from the MITRE ATT&CK procedures corpus using the implementation of the class TfidfVectorizer from Skitlearn, as described in section III-A7. The TfidfVectorizer class has a parameter named min\_df which is used to inform the algorithm of a threshold to ignore terms when building the vocabulary that has a document frequency strictly lower than it. The default parameter for min\_df is 1, which means ignoring terms that appear in less than 1 document.

 **FIGURE . Labelling Tweets using Doccano - sample data**

**TABLE 12. One-Class results for min\_df variations from 1 to 4 for tweets classification.**



However, in this experiment, we increased this number to make the algorithm ignore more infrequent terms while building the One-class training set to check its influence on classification performance. Before we pass the tweet messages to the One-class classifier, we must transform the tweets into a document-term matrix using the vocabulary and the term frequency (TF) produced in the One-class training phase. To do this, we used the method transform from TfidfVectorizer.

The results for the One-Class classification of 241 labeled tweets for ‘min\_df’ ranging from 1 to 4 are shown in 12.

Our experiments showed that the higher the value of min\_df, the lower the number of selected tweets in this stage of the pipeline. For min\_df of 1, there were selected 78, for min\_df of 2, 40 tweets, for min\_df of 3, 29 tweets, and for min\_df of 4, 28 tweets.

The best accuracy was 78.42% when the ‘min\_df’ value is set to 1. On the other hand, the precision, which means the proportion of positive identifications actually correct, is better for ‘min\_df’ 2 and 3, with values of 85% and 86.20% respectively.

D. MULTI-CLASS CLASSIFICATION This step of the pipeline consists in identifying the MITRE ATT&CK tactic most likely described in each tweet message considered normal by the One-Class classifier run in the previous step of the pipeline.

As described in Section III-A10 the multi-class model consists of 14 classes trained with the MITRE ATT&CK’s procedures grouped by the tactic. Similarly to what was performed for the One-class classifier, the training data for the multi-class consists of a TF-IDF feature matrix created from the MITRE ATT&CK procedures. The difference is that for the multi-class the procedure corpus was divided into 14 tactics, one for each class.

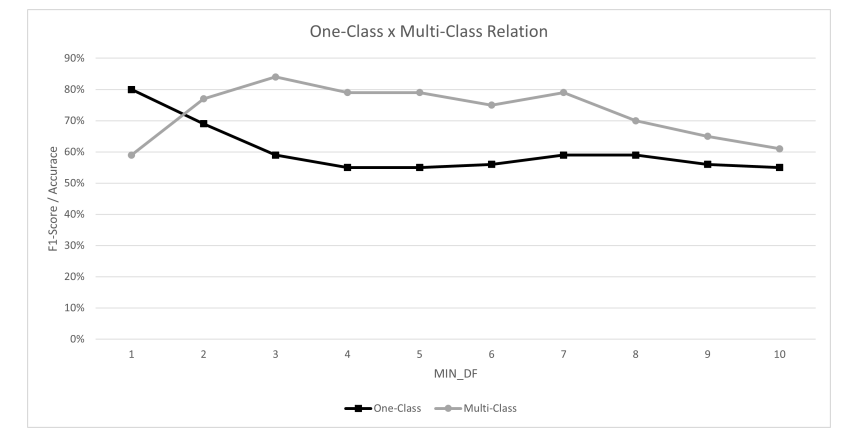
To implement the classifier, we used the Linear SVC (Support Vector Classification) implementation from Skitlearn.24.

To compare the results, we executed the Multi-Class step for each One-Class result. As a reminder, the One-Class results were processed using values from 1 to 4 for the min\_df parameter.

Multi-class results are presented in Tables 13, 14, 15 and 16.

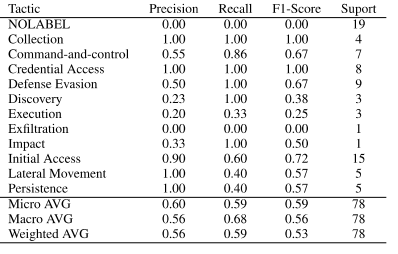
We performed experiments with higher values for min\_df until 10 to check the influence of this parameter in the F1- score performance for results for both One-class and Multiclass classifiers. The results are shown in Figure 6.

Given the results comparing different values for min\_df parameter, the higher the min\_df parameter until the limit of 3, we noticed that:



**FIGURE. One-Class and Multi-Class relation in terms of min\_df parameter.**

**TABLE 13. Multi-class results for One-class min\_df=1**



• the lower is the number of tweet messages resulting from one-class classifier and automatically the lower is the number of input tweets for the multi-class classifier;

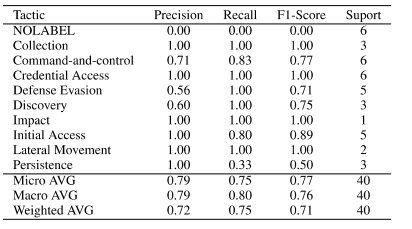
• the lower is the accuracy of one-class classifier;

• the higher is the false negative of one-class classifier;

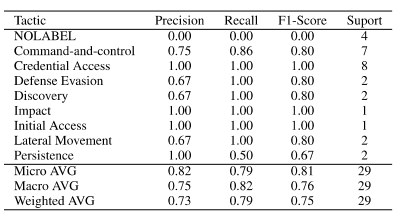
• the higher is the precision of the one-class classifier;

• the higher is the F1-Score of the multi-class classifier. From this, we can conclude that the min\_df parameter is turning the one-class classifier more restrictive, making it select tweet messages closest to the description of a malicious behavior described in MITRE ATT&CK procedures. This is reflected in better F1-Scores in multi-class classifier. The downside of having a more restrictive one-class is the increase of false negatives for the one-class. This means that we mistakenly discard tweets that should not be discarded.

**TABLE 14. Multi-class results for One-class min\_df=2**

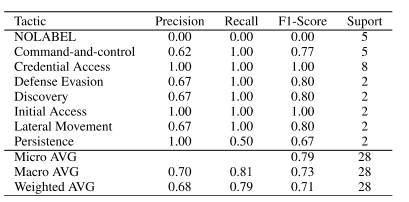


**TABLE 15. Multi-class results for One-class min\_df=3**

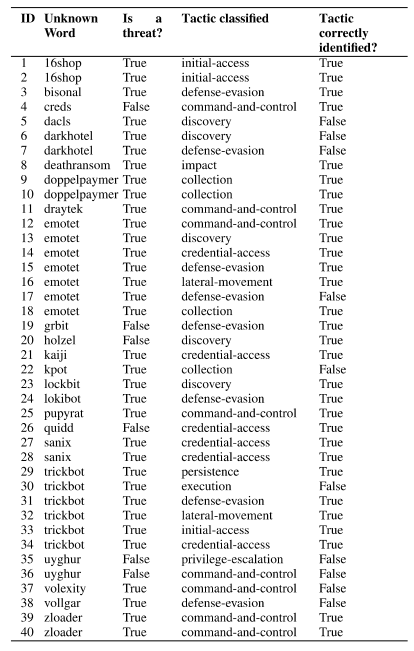


The balance value for the min\_df value can be interpreted this way: if your system is more important to identify new threats than the accuracy of the respective tactics, a value of 1 is suitable for min\_df. If the accuracy of identifying the respective tactics for the identified threats is more important than the number of threats, a min\_df of 3 is most appropriate. In our experiment, as seen in Figure 6, a good trade-off has been achieved using a min\_df = 2. With this, we had 69% for One-Class and 77% for the Multi-Class. E. ALERTS This is the last step of the pipeline. It consists of generating alerts to cyber threat analysts regarding identified threats and corresponding id.

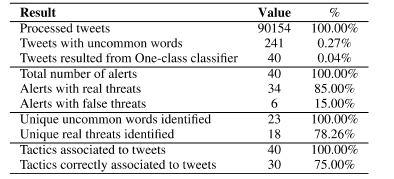
**TABLE 16. Multi-class results for One-class min\_df=4.**



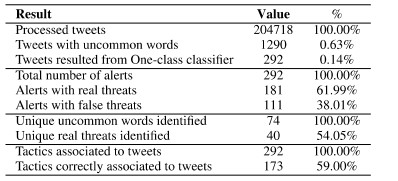
**TABLE 17. Alert list**



**TABLE 18. Experiment summary.**



**TABLE 19. Execution summary**



The input for the alert generation procedure is the output of the multi-class classification. As described in the previous section, we executed the multi-class step using different values for the variable min\_df. However, the results we are going to present in this section are for the min\_df of 2, which resulted in 40 classified tweets. The alert procedure generated a total of 40 alerts detailed in Table 17. From those, 34 were alerts for repeal threats and a total of 30 tactics were correctly identified. The Table 18 summarizes the results of the entire experiment.

# 

# CHAPTER 9:CONCLUSION

# Given the dynamism of the cyber security field, with new vulnerabilities and threats appearing at any time, keeping up to date on them is a challenging but important task for analysts. Even following the best practices and applying the best controls, a new threat may bring an unusual way to subvert the defenses requiring a quick response. This way, timely information about emerging cyber threats becomes paramount to a complete cyber security system.

# This research proposes an automated cyber threat identification and profiling based on the natural language processing of Twitter messages. The objective is exactly to cooperate with the hard work of following the rich source of information that is Twitter to extract valuable information about emerging threats in a timely manner.

# This work differentiates itself from others by going a step beyond identifying the threat. It seeks to identify the goals of the threat by mapping the text from tweets to the procedures conducted by real threats described in MITRE ATT&CK knowledge base. Taking advantage of this evolving and collaborative knowledge base to train machine learning algorithms is a way to leverage the efforts of cyber security community to automatically profile identified cyber threats in terms of their intents.

# To put in test our approach, in addition to the research experiment, we implemented the proposed pipeline and run it for 70 days generating online alerts for the Threat Intelligence Team of a big financial institution in Brazil. During this period, at least three threats made the team take preventive actions, such as the Petit Potam case, described in section V. Our system alerted the team making them aware of Petit- Potam 17 days before the official patch was published by Microsoft. Within this period, the defense team was able to implement mitigations avoiding potential exploits and, consequently, incidents.

# Our experiments showed that the profiling stage reached an F1 score of 77% in correctly profiling discovered threats among 14 different tactics and the percentage of false alerts of 15%. In future work, we consider it important to advance in tweets selection stages (Unknown Words and One-class), to improve the false positives rate and in the profiling stage, to reach higher accuracy in determining the technique associated with the identified threat. We are working on this way by experimenting with a different NLP approach using the part of speech (POS) algorithm implementation from Spacy29 Python library. The object is to identify the root verb, the subject, and the object of the phrases to select tweets where the action described (the root verb) is referencing the unknown word (the subject).

# CHAPTER-10:BIBLIOGRAPHY

[1] B. D. Le, G. Wang, M. Nasim, and A. Babar, ‘‘Gathering cyber

threat intelligence from Twitter using novelty classification,’’ 2019,

*arXiv:1907.01755*.

[2] *Definition: Threat Intelligence*, Gartner Research, Stamford, CO, USA,

2013.

[3] R. D. Steele, ‘‘Open source intelligence: What is it? why is it important to

the military,’’ *Journal*, vol. 17, no. 1, pp. 35–41, 1996.

[4] C. Sabottke, O. Suciu, and T. Dumitras, ‘‘Vulnerability disclosure in the age

of social media: Exploiting Twitter for predicting real-world exploits,’’ in

*Proc. 24th USENIX Secur. Symp. (USENIX Secur.)*, 2015, pp. 1041–1056.

[5] A. Sapienza, A. Bessi, S. Damodaran, P. Shakarian, K. Lerman, and

E. Ferrara, ‘‘Early warnings of cyber threats in online discussions,’’ in

*Proc. IEEE Int. Conf. Data Mining Workshops (ICDMW)*, Nov. 2017,

pp. 667–674.

[6] E. Nunes, A. Diab, A. Gunn, E. Marin, V. Mishra, V. Paliath, J. Robertson,

J. Shakarian, A. Thart, and P. Shakarian, ‘‘Darknet and deepnet mining

for proactive cybersecurity threat intelligence,’’ in *Proc. IEEE Conf. Intell.*

*Secur. Informat. (ISI)*, Sep. 2016, pp. 7–12.

[7] S. Mittal, P. K. Das, V. Mulwad, A. Joshi, and T. Finin, ‘‘CyberTwitter:

Using Twitter to generate alerts for cybersecurity threats and vulnerabilities,’’

in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining*

*(ASONAM)*, Aug. 2016, pp. 860–867.

[8] A. Attarwala, **S.** Dimitrov, and A. Obeidi, ‘‘How efficient is Twitter:

Predicting 2012 U.S. presidential elections using support vector machine

via Twitter and comparing **against** Iowaelectronicmarkets,’’ inProc***.*** *Intell.*

*Syst. Conf. (IntelliSys)*, Sep. 2017, pp. 646–652.

[9] N. Dionísio, F. Alves, P. M. Ferreira, and A. Bessani, ‘‘Towards end-to-end

cyberthreat detection from Twitter using multi-task learning,’’ in *Proc. Int.*

*Joint Conf. Neural Netw. (IJCNN)*, Jul. 2020, pp. 1–8. [10] O. Oh, M. Agrawal, and H. R. Rao, ‘‘Information control and terrorism:

Tracking the Mumbai terrorist attack through Twitter,’’ *Inf. Syst. Frontiers*,

vol. 13, no. 1, pp. 33–43, Mar. 2011.

[11] T. Sakaki, M. Okazaki, and Y. Matsuo, ‘‘Earthquake shakes Twitter users:

Real-time event detection by social sensors,’’ in *Proc. 19th Int. Conf.World*

*Wide Web*, Apr. 2010, pp. 851–860.

[12] B. De Longueville, R. S. Smith, and G. Luraschi, ‘‘‘OMG, from here, i

can see the flames!’: A use case of mining location based social networks

to acquire spatio-temporal data on forest fires,’’ in *Proc. Int. Workshop*

*Location Based Social Netw.*, Nov. 2009, pp. 73–80.

[13] A. Sapienza, S. K. Ernala, A. Bessi, K. Lerman, and E. Ferrara, ‘‘DISCOVER:

Mining online chatter for emerging cyber threats,’’ in *Proc.*

*Companion Web Conf. Web Conf. (WWW)*, 2018, pp. 983–990.

[14] R. P. Khandpur, T. Ji, S. Jan, G. Wang, C.-T. Lu, and N. Ramakrishnan,

‘‘Crowdsourcing cybersecurity: Cyber attack detection using

social media,’’ in *Proc. ACM Conf. Inf. Knowl. Manage.*, Nov. 2017,

pp. 1049–1057.

[15] Q. Le Sceller, E. B. Karbab, M. Debbabi, and F. Iqbal, ‘‘SONAR: Automatic

detection of cyber security events over the Twitter stream,’’ in *Proc.*

*12th Int. Conf. Availability, Rel. Secur.*, Aug. 2017, pp. 1–11.

[16] K.-C. Lee, C.-H. Hsieh, L.-J.Wei, C.-H. Mao, J.-H. Dai, and Y.-T. Kuang,

‘‘Sec-buzzer: Cyber security emerging topic mining with open threat intelligence

retrieval and timeline event annotation,’’ *Soft Comput.*, vol. 21,

no. 11, pp. 2883–2896, Jun. 2017.

[17] A. Ritter, E. Wright, W. Casey, and T. Mitchell, ‘‘Weakly supervised

extraction of computer security events from Twitter,’’ in *Proc. 24th Int.*

*Conf. World Wide Web*, May 2015, pp. 896–905.

[18] A. Queiroz, B. Keegan, and F. Mtenzi, ‘‘Predicting software vulnerability

using security discussion in social media,’’ in *Proc. Eur. Conf. Cyber*

*Warfare Secur.*, 2017, pp. 628–634.

[19] A. Bose, V. Behzadan, C. Aguirre, and W. H. Hsu, ‘‘A novel approach for

detection and ranking of trendy and emerging cyber threat events in Twitter

streams,’’ in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining*

*(ASONAM)*, Aug. 2019, pp. 871–878.

[20] B. E. Strom, A. Applebaum, D. P. Miller, K. C. Nickels, A. G. Pennington,

and C. B. Thomas, ‘‘Mitre ATT&CK: Design and philosophy,’’ MITRE

Corp., McLean, VA, USA, Tech. Rep. 19-01075-28, 2018.

[21] B.-J. Koops, J.-H. Hoepman, and R. Leenes, ‘‘Open-source intelligence

and privacy by design,’’ *Comput. Law Secur. Rev.*, vol. 29, no. 6,

pp. 676–688, Dec. 2013.

[22] R. Campiolo, L. A. F. Santos, D. M. Batista, and M. A. Gerosa, ‘‘Evaluating

the utilization of Twitter messages as a source of security alerts,’’ in *Proc.*

*28th Annu. ACM Symp. Appl. Comput.*, Mar. 2013, pp. 942–943.

[23] N. Dionísio, F. Alves, P. M. Ferreira, and A. Bessani, ‘‘Cyberthreat detection

from Twitter using deep neural networks,’’ in *Proc. Int. Joint Conf.*

*Neural Netw. (IJCNN)*, Jul. 2019, pp. 1–8.

[24] A. Niakanlahiji, J.Wei, and B. Chu, ‘‘A natural language processing based

trend analysis of advanced persistent threat techniques,’’ in *Proc. IEEE Int.*

*Conf. Big Data (Big Data)*, Dec. 2018, pp. 2995–3000.

[25] G. Ayoade, S. Chandra, L. Khan, K. Hamlen, and B. Thuraisingham,

‘‘Automated threat report classification over multi-source data,’’ in *Proc.*

*IEEE 4th Int. Conf. Collaboration Internet Comput. (CIC)*, Oct. 2018,

pp. 236–245.