Cyber Threat Predictive Analytics for Improving Cyber Supply Chain (CSC) Security

A Project report submited to

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTHAPUR

In partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

V. SULOCHANA	188X1A05B3
S. BHAVANA	188X1A0584
R. HEMASREE	188X1A0583
N. NANDHINI	188X1A0569
M. KEERTHI	188X1A0565

Under the Esteemed Guidance of

Mrs.C.MADHURI., M.TECH.,

Assistant Professor of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MJR COLLEGE OF ENGINEERING & TECHNOLOGY

(Approved by A.I.C.T.E, New Delhi and Affiliated to J.N.T.U.A.Ananthapuramu)

Diguvapokula vari palli, Pulicherla mandal, Chittoor district-517113.

(2018-2022)

MJRA COLLEGE OF ENGINEERING & TECHNOLOGY

(Approved by A.I.C.T.E, New Delhi and Affiliated to J.N.T.U.A.Ananthapuramu)

Diguvapokula vari palli, Pulicherla mandal, Chittoor district-517113.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that the project work entitled "**Cyber Threat Predictive Analytics for Improving Cyber Supply Chain Security**" is a bonafide work done by

V. SULOCHANA	188X1A05B3
S. BHAVANA	188X1A0584
R. HEMASREE	188X1A0583
N. NANDHINI	188X1A0569
M. KEERTHI	188X1A0565

In partial fulfillment of requirements for the award of degree Bachelor of Technology in Computer Science and Engineering during the academic year 2018-2022.

PROJECT GUIDE	HEAD OF THE DEPARTMENT
Mrs.C.MADHURI.,M.TECH.,	Dr.A.SATHIYARAJ.,B.TECH.,ME.,.
Assistant Professor, Dept of CSE.	M.B.A.,PhD.,Professor, Dept of CSE.

Submitted for University Examination (Viva Voice) held on _____.

INTERNAL EXAMINER. EXTERNAL EXAMINER.

DECLARATION

We are here by declare that project report entitled "**Cyber Threat Predictive Analytics** Bachelor of Technology in Computer Science & Engineering, **JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY Ananthapuramu,** is a bonafide record of the **MJR COLLEGE OF ENGINEERING & TECHNOLOGY** and has not been submitted to any other courses or university for award of any Degree.

V. SULOCHANA	188X1A05B3
S. BHAVANA	188X1A0584
R. HEMASREE	188X1A0583
N. NANDHINI	188X1A0569
M. KEERTHI	188X1A0565

DATE:

PLACE:

ACKNOWLEDGEMENT

The completion of project brings with and sense of satisfaction, but it is never completed without thinking the persons who are all responsible for its successful completion. I wish to express my profound feelings of gratitude to this great institution of ours **MJR COLLEGE OF ENGINEERING AND TECHNOLOGY** for providing us the excellent facilities.

I wish to express my sincere thanks to our chairman **Sri M.P. AVINASH KUMAR REDDY** Piler, Chittoor Dist, our beloved vice chairman for having provided all the facilities and support in completing my project successfully.

I earnestly thank **Dr. N. SUDHAKAR REDDY, M.TECH., Ph.D.,** Principal for having permitted me to carry out the project on **"Cyber Threat Predictive Analytics for Improving Cyber Supply Chain Security"** successfully.

I extremely thankful to **Head Of the Department Dr. A. SATHIYARAJ, B. Tech,. M.E., M.B.A., PhD., Professor, Computer Science and Engineering,** for the valuable support given to me, without whom we would not have been able to complete my project work.

I wish to express my deep sense of gratitude to internal guide., **Mrs. C. MADHURI., M.TECH., Assistant Professor,** Department of Computer Science and Engineering for his valuable guidance and useful suggestions at all the stages of the project, which helped me in completing the project work in time.

Nevertheless, I express my appreciation to the staff, one and all whose creative ideas and suggestions helped me in completing this project.

CONTEXT

ABSTRACT

The Cyber Supply Chain (CSC) system is complex which involves different subsystems performing various tasks. Security in the supply chain is challenging due to the inherent vulnerabilities and threats from any part of the system which can be exploited at any point within the supply chain. This can cause a severe disruption on the overall business continuity. Therefore, it is paramount important to understand and predicate the threats so that organizations can undertake necessary control measures for the supply chain security. Cyber Threat Intelligence (CTI) provides an intelligence analysis to discover unknown to known threats using various properties including threat actor skill and motivation, Tactics, Techniques, and Procedure (TT and P), and Indicator of Compromise (IoC). This paper aims to analyze and predicate threats to improve cyber supply chain security. We have applied Cyber Threat Intelligence (CTI) with Machine Learning (ML) techniques to analyze and predict the threats based on the CTI properties. That allows to identify the inherent CSC vulnerabilities so that appropriate control actions can be undertaken for the overall cybersecurity improvement. To demonstrate the applicability of our approach, CTI data is gathered and a number of ML algorithms, i.e., Logistic Regression (LG), Support Vector Machine (SVM), Random Forest (RF), and Decision Tree (DT), are used to develop predictive analytics using the Microsoft Malware Prediction dataset. The experiment considers attack and TTP as input parameters and vulnerabilities and Indicators of compromise (IoC) as output parameters. The results relating to the prediction reveal that Spyware/Ransomware and spear phishing are the most predictable threats in CSC. We have also recommended relevant controls to tackle these threats. We advocate using CTI data for the ML predicate model for the overall CSC cyber security improvement.

LIST OF FIGURES

LIST OF SCREEN SHOTS

LIST OF TABLES

CHAPTER-1
INTRODUCTION

1. INTRODUCTION

1.1 MOTIVATION

Cyber Supply Chain (CSC) security is critical for reliable service delivery and ensure overall business continuity of Smart CPS. CSC systems by its inherently is complex and vulnerabilities within CSC system environment can cascade from a source node to a number of target nodes of the overall cyber physical system (CPS). A recent NCSC report highlights a list of CSC attacks by exploiting vulnerabilities that exist within the systems [1]. Organizations outsource part of their business and data to the third-party service providers that could lead any potential threat. There are several examples for successful CSC attacks. For instance, Dragonfly, a Cyber Espionage group, is well known for targeting CSC organization [2], [3]. The Saudi Aramco power station attack halted its operation due to a massive cyberattack [1]. There are existing works that consider CSC threats and risks but a lack of focus on threat intelligence properties for the overall cyber security improvement

1.2 PROBLEM DEFINITION

Further, it is also essential to predict the cyberattack trends so that the organization can take the timely decision for its countermeasure. Predictive analytics not only provide an understanding of the TTPs, motives and intents of the threat actors but also assist situational awareness of current supply system vulnerabilities. This paper aims to improve the cybersecurity of CSC by specifically focusing on integrating Cyber Threat Intelligence (CTI) and Machine Learning (ML) techniques to predicate cyberattack patterns on CSC systems and recommend suitable controls to tackle the attacks.

1.3 OBJECTIVE OF PROJECT

Firstly, we consider Cyber Threat Intelligence (CTI) for systematic gathering and analysis of information about the threat actor and cyber-attack by using various concepts such as threat actor skill, motivation, IoC, TTP and incidents. The reason for considering CTI is that it provides evidence-based knowledge relating to the known attacks.

This information is further used to discover unknown attacks so that threats can be well understood and mitigated. CTI provides intelligence information with the aim of preventing attacks as well as shorten time to discover new attacks.

Secondly, we applied ML techniques and classification algorithms and mapped with the CTI properties to predict the attacks. We use several classification algorithms such as Logistic Regression (LG), Support Vector Machine (SVM), Random Forest (RF) and Decision Tree (DT) for this purpose. We follow CTI properties such as Indicator of Compromise (IoC) and Tactics, Techniques and Procedure (TTP) for the attack predication.

Finally, we consider widely used cyberattack dataset to predict the potential attacks. The predication focuses on determining threats relating to Advance Persistent Threat (APT), command and control and industrial espionage which are relevant for CSC. The result shows the integration of CTI and ML techniques can effectively be used to predict cyberattacks and identification of CSC systems vulnerabilities. Furthermore, our prediction reveals a total accuracy of 85% for the TPR and FPR. The results also indicate that LG and SVM produced the highest accuracy in terms of threat predication

The rest of the paper is organised as follows: Section 2 presents an overview of related works including CSC security, cyber threat intelligence and Machine Learning for CSC. Section 3 provides the concepts necessary for the proposed approach and the meta model. Section 4 provides an overview of the proposed approach including the integration of CTI and ML. Section 5 presents the underlying process for the threat analysis and predication. Section 6 implements the process for the threat predication using the widely used Microsoft malware datasets. Section 7 discusses the results and compares the work with the existing works in the literature. Finally, Section 8 provides conclusion and future direction of the work.

CHAPTER-2

LITERATURE SURVEY

2. LITERATURE SURVEY

2.1 INTRODUCTION

There exists several widely used CTI and ML models in cyber security domain. This section presents the existing works that are relevant with our work.

2.2 CYBER SUPPLY CHAIN (CSC) SECURITY

The CSC security provides a secure integrated platform for the inbound and outbound supply chains systems with third party service provider including suppliers, and distributors to achieve the organizational goal. Cybersecurity from supply chain context involves various secure outsourcing of products and information between third party vendors, and suppliers. This outsourcing includes the integration of operational technologies (OT) and Information technologies (IT) running on Cyber Physical Systems (CPS) infrastructures. However, there are threats, risks and vulnerabilities that are inherent in such systems that could be exploited by threat actors on the operational technologies and information technologies of the supply inbound and outbound chains systems. The outbound chain attacks include data manipulations, information tampering, redirecting product delivery channels, and data theft. The IT risks include those attacks on the cyber physical and cyber digital system components such as distributed denial of service (DDoS) attacks, IP address spoofing, and Software errors. Regarding CSC security, NIST SP800 proposed a 4 tier framework approach for improving critical infrastructure cybersecurity that incorporates the cyber supply chain risk management framework into it as one of its core components. Tier 1 considers the organizations CSC risk requirement strategy. Tier 2 considers the supply chain associated risk identifications including products and services in the supply inbound and outbound chains. Tier 3 implementation considers the risk assessments, threats analyses, associated impacts and determine the baseline requirements for governance structure. Tier 4 consider real-time or near-time information to understand supply chain risk associated with each product and service. However, the approach and tiers considered risks management but did not emphasize on ML and threat prediction for future trends in the CSC domain. Additionally, proposed a supply chain attack framework and attack patterns that structured and codifies supply chain attacks.

The goal of the framework was to provide a comprehensive view of supply chain attacks of malicious insertion across the full acquisition lifecycle to determine the associated threat and vulnerability information.

2.3 CYBER THREAT INTELLIGENCE (CTI)

Cyber threat intelligence (CTI) gatherings and analysis have become one of the relevant actionable intelligences used to understand both known and unknown threats. The impact of cyberattacks and emerging threats on CSC systems and its devastating effects on business process, data, Intellectual Property, delivery channel, and cost of recovery has increased the surge for CTI approach. The CTI process includes identification, threat analysis and information disseminating to stakeholders. Considering CTI for cybersecurity, ENISA in explored the opportunities and limitations of current threat intelligence platforms by considering CTI implementation process and threat intelligence programs (TIP) from strategic, tactical and operational goals. The authors proposed a threat intelligence program model that collects, normalize, enrich, correlate, analyse and disseminate threat related information to stakeholders. The strategic CTI goals consider factors that support executive decision makings, tactical goals consider the CTI process and TIP programs that identifying intelligence gap and prioritizing them for risk reduction. The operational goals provide a process that provides an understanding of the threat actors motives, modes of operation, intents, and TTPs and capabilities. However, the processes do not incorporate ML threat predictions. Additionally, proposes a threat intelligence-driven security model that considers six CTI phases and processes lifecycle required to identify intelligence goals. The CTI phases include direction, collection, process, analysis, dissemination, and feedback. The author incorporated internal sources such as network traffic, logs, scans; external sources such as vulnerability database, threat feeds; and human sources such as the dark web and social media into the model for the threat intelligence modelling. The threat intelligence driven security model emphasizes on using network traffics, logs and scans and not ML algorithms for the prediction. Further, develop cyber threat Intelligence metrics that consider assets, requirement business operations, adversary, and consumer intelligence places emphases on value and organizational benefits.

The author's approach considers four key stages in the threat intelligence process including intelligence requirements, information collection, analyses, dissemination, and intelligence usage. However, the approach does not consider machine learning for predicting invisible attacks.

Furthermore, proposed a CTI model that operationalizes and analyses adversarial activities across the lifecycle of an organization business process to determine actions taken by the attacker. The author's approach was based on the organizational intelligence requirements, information gathering, analyses and disseminate to protect assets for strategic, tactical and operational understanding and situational awareness. However, the works emphasized more on attacker motive and intent and not on ML for the threat predictions. The CTI functional process is to collect metrics and trend analysis for the business risk assessment, prioritization, and decision support with less emphasis on ML for CSC security

2.4 MACHINE LEARNING IN CSC SECURITY

There are several works that consider Machine Learning classifiers in various cybersecurity application domains such as spam filters, antivirus and IDS/IPS to predict cyberattack trends. Considering ML for Security, proposed ML classification of HTTP attacks using a decision tree algorithm to learn a dataset for performance accuracies and automatically label a request as valid or attack. The authors developed a vector space model used commonly for information retrieval to build a classifier to automatically label the request as malicious in the URL. The approach achieved high precision and recall comparatively. However, the work did not focus on ML and threat prediction in the CSC environment. Further, carried out the feasibility of a study on machine learning models for cloud security to test the models in diverse operation conditions cloud scenarios.

The authors compared Logistic Regression, Decision Tree, Naïve Bayes, and SVM classification algorithms techniques to learn a dataset for performance accuracies. The algorithms represent supervised schemes and are used in network security. The result shows an accuracy of 97% in anomalous packet detections. However, the work did consider CSC security from threat prediction in the supply chain environment. Furthermore, surveyed data mining and ML methods for cybersecurity detection methods for cyber analytics in support of intrusion detection in cybersecurity applications.

The authors used Artificial Neural Network, Association rules, Fuzzy Association rules and Bayesian Networks classifiers to learn the datasets and provided comparison criteria for the machine learning and data mining models to recognize the types of the attack (misuse) and for detection of an attack (intrusion). However, the techniques and methods used are not ML models and did not focus on ML and threat prediction in the CSC environment.

Additionally, review the cybersecurity dataset for ML algorithms used for analysing network traffic and anomaly detection. The author compared the machine learning techniques used for experiments, evaluation methods and baseline classifiers for comparison of the dataset. The results show significant flaws in some dataset during feature selection and are not relevant for modern intrusion detections datasets. However, the review did not stress on the current dataset we used from the Microsoft Malware Threat Prediction website for the prediction. Moreover, explored the classification of logs using ML techniques on a decision tree algorithm to learn a dataset that models the correlation and normalization of security logs. The goal of the ML techniques is to evaluate if the algorithm can predict the performance of classification as an attack or not after a training phase. The dataset used contains anomalous and some identified attacks. The result shows that the DT algorithm was model on internet logs to develop a framework for the normalization and correlation of the classify with an accuracy of 80%. However, the classification model did not compare other classification algorithms such as SVM, LR and RF that are relevant for ML better performance accuracies and threat analysis.

Another initiative explores the viability of using machine learning approaches to predict power systems disturbance and cyberattack discrimination classifiers and focuses specifically on detecting cyberattacks where deception is the core tenet of the event. The authors in evaluated the classification performances on, NNge, OneR, SVM, RF, JRpper and Adaboost algorithms to learn the dataset and focused specifically on detecting cyber attacks where deception is the core tenet of the event. For example, in, the authors proposed a SCADA power system cyberattack detection approach by combining a correlation-based feature selection (CFS) method and K-Nearest-Neighbour (KNN) instance-based learning (IBL) algorithm.

The combination was useful to reduce the extremely large number of features and to maximize cyberattack detection accuracy with minimum detection time cost.

In, an ensemble-learning model for detecting the cyberattacks of SCADA-based IoT platform is proposed. The model was based on the combination of a random subspace (RS) learning method with random tree (RT). The authors in proposed a deep-learning, feature-extraction-based semi-supervised model for cyberattack protection in the trust boundary of IoT networks. The proposed approach was adaptive to learn unknown attack.

However, the works did not consider CSC attacks from supplier inbound and outbound chains.

Regarding ML predictive analytics on various datasets, predicted cybersecurity incidents using ML algorithms to distinguish between the different types of models. The authors used text mining methods such as n-gram, bag of-words and ML techniques to learn dataset on Naive Bayes and SVM algorithms for classification performance. The experiment was to predict classification accuracies of malware incidents response and actions. The approach did not consider CTI and ML in the CSC system environment. Further, proposed a risk teller system that analyses binary file appearance logs of a machine to predict which machines are at risk of experiencing malware infection in advance. The authors used a random forest algorithm and semi-quantitative methods to build a risk prediction model that creates a profile to capture usage patterns. The results associate each level of risk to a machine infection incident with 95% true positive precision. Besides, characterize the extent to which cyber security incidents can be predicted based on externally observable properties of an organization’s network.

Therefore, it is necessary to use ML analytics to predict cyber-attacks, threats and the underlying vulnerabilities. Additionally, there is a need to understand an organisational context for the threat analysis. CTI can effectively support to achieve that goal. This work contributes towards this direction. We have integrated CTI for threat gathering and analysis with the ML for the threat prediction so that organizations can determine the suitable control measure for the overall CSC security improvement.

CHAPTER-3

SYSTEM ANALYSIS

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Cyber Supply Chain (CSC) security is critical for reliable service delivery and ensures overall business continuity of Smart CPS. A recent NCSC report highlights a list of CSC attacks by exploiting vulnerabilities that exist within the systems.

Organizations outsource part of their business and data to the third-party service providers that could lead to any potential threat. There are several examples of successful CSC attacks. For instance, Dragonfly, a Cyber Espionage group, is well known for targeting CSC organizations. The Saudi Aramco power station attack halted its operation due to a massive cyberattack. There are existing works that consider CSC threats and risks but a lack of focus on threat intelligence properties for the overall cyber security improvement.

DISADVANTAGES

- Existing works that consider CSC threats and risks but a lack of focus on threat intelligence properties for the overall cyber security improvement.
- A limited works emphasize on threat intelligence data for the attack prediction

3.2 PROPOSED SYSTEM

This section discusses the proposed approach that aims to improve the CSC security. It includes an integration of CTI and ML and a systematic process (presented in the Section 5). Additionally, the underlying concepts of the proposed approach such as actor, goal, TTP, vulnerability, incident, and controls, is also mentioned in Section 3. The approach considers both inbound and outbound chains for the vulnerability so that CSC organisation can focus on the possible system flaws. The approach adopts the CTI process to gather and analyse the threat data and ML techniques to predicate the threat. ML techniques are used on classification algorithms to learn a dataset for performance accuracies and predictive analytics. The rationale for integrating CTI and ML for threat prediction is that the CTI lifecycle process supports input parameters for detecting known attacks whereas ML provides output parameters for predicting known and unknown attacks for future trends.

INTEGRATION OF CTI AND ML

The approach combines CTI processes with ML techniques for cyber threat predictive analytics. The goal is to detect vulnerabilities and indicators of compromise on CSC network system nodes using known attacks to predict unknown attacks. We apply the CTI techniques to gather threats (Known attacks) and ML techniques to learn the dataset to predicate cyber threats (unknown attacks) on CSC systems.

The inputs are the attacks and TTP that are deployed by threat actors to compromise a system. The attack feature uses properties such as attack type, pattern, attack vectors, and prerequisites to determine the nature of the attack that was deployed. The TTP consists of attack patterns and attack vectors deployed by the threat actor. The TTP parameter includes the capabilities of the threat actor and threat indicators. The threat actor feature uses properties such as user, system and third-party vendors to determine the vulnerable spots and type of tools used for the attack to determine the attack pattern.

Tools are the attack weapons or software codes used by the threat actor for reconnaissance and to initiate an attack. For instance, the threat actor could use Nmap tool for scanning a network, Kali Linux tool for penetration and, Metasploit tool for exploiting loopholes in a network. The output parameters are the vulnerabilities and indicators of compromise that are used as threat intelligence. The capability of the threat actor could be determined by the ability to penetrate a system and course Advance Persistent threat (APT) attack and take command and control C&C) the extent of propagation is used to determine the indicators.

Finally, we consider various controls such as directive, preventive, detective corrective and recovery required to secure the CSC system. The rationale for our predictive analytics approach is based on the premise that the cyberattacks phenomenon includes a lot of invincibility, and uncertainties and the makes the threat landscape unpredictable. Similarly, due to the changing organizational requirements, various integrations, varying business processes and the various delivery mechanisms,

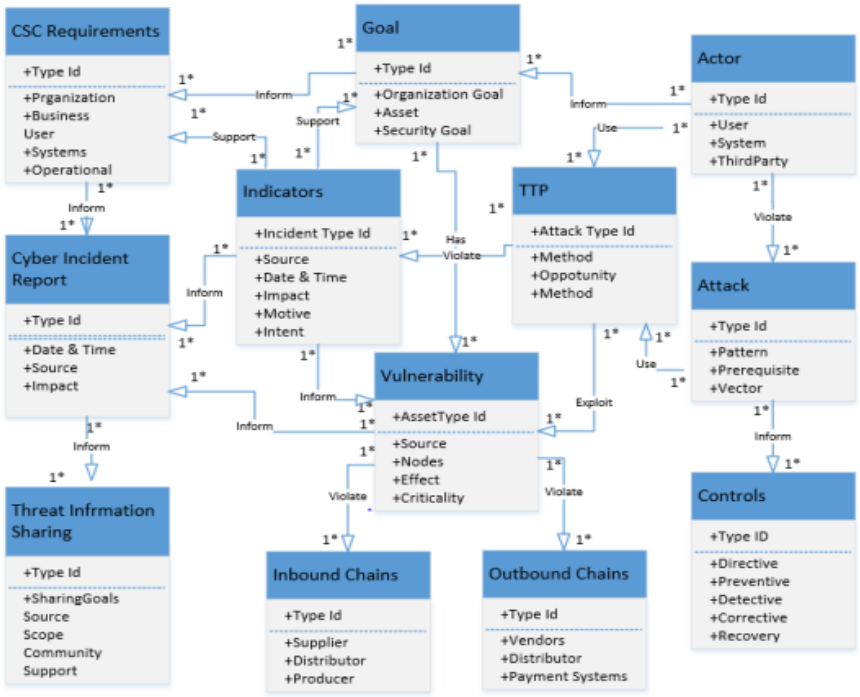


FIGURE 1. Meta-model for the proposed conceptual view of CSC system security

predicting cyberattacks in the CSC organization context has been challenging. To achieve that, first, the proposed approach considers relevant related works and the metamodel concepts to model the CSC attacks and CTI phases. For instance, we identify supply inbound and outbound chain attack indicators and integrate them into CTI phases. Further, the concepts are analysed using the CTI process lifecycle and ML techniques to learn the dataset for our prediction. Furthermore, we use the input and output parameters as indicators for our threat prediction. Finally, the threat prediction results are evaluated to provide informed intelligence regarding the various attacks and future threats that are unknown for appropriate control mechanisms. Figure 2 indicates the proposed approach.

ADVANTAGES

- CTI provides intelligence information with the aim of preventing attacks as well as shorten time to discover new attacks.
- The result shows the integration of CTI and ML techniques can effectively be used to predict cyberattacks.

3.3 ALGORITHM DESCRIPTION

IMPLEMENTATION ALGORITHM

❖ NAÏVE BAYE'S ALGORITHM

- Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.
- For some types of probability models, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood.

❖ SUPPORT VECTOR MACHINE

- In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. SVM maps training examples to points in space so as to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

- In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

❖ LOGISTIC REGRESSION

- Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labelled "0" and "1".
- In the logistic model, the log-odds (the logarithm of the odds) for the value labeled "1" is a linear combination of one or more independent variables ("predictors"); the independent variables can each be a binary variable (two classes, coded by an indicator variable) or a continuous variable (any real value). The corresponding probability of the value labeled "1" can vary between 0 (certainly the value "0") and 1 (certainly the value "1"), hence the labeling; the function that converts log-odds to probability is the logistic function, hence the name.

❖ RANDOM FOREST

- Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.
- As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

3.4 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

- Technical Feasibility
- Economic Feasibility
- Operational Feasibility

TECHNICAL FEASIBILITY

This project, Data Centric Knowledge Management System needs the support to web based technology being implemented for other useful systems in our company. It requires PC's and NIC Card with normal configuration for Intranet access. Almost all administrators have their own PC on their desk. Thus, it is technically feasible to implement the new system here.

ECONOMIC FEASIBILITY

"Data Centric Knowledge Management" is an in-house project. It is very much useful for the company to maintain their knowledge assets. The infrastructure for the development of their new system is available in the campus itself.

The system is developed at no additional cost. Hence it is economically feasible for the new system to be implemented.

OPERATIONAL FEASIBILITY

This system is being automated on the request of the technical department of our company. This new system meets their requirement and covers all aspects required much better than the old manual system. Most of the people involved in this branch are computer literate and do not need much training if this system is implemented. Hence it is operationally feasible.

3.5 FUNCTIONAL REQUIREMENTS

- Data owners encrypt their data files and store them in the cloud for sharing with data consumers.
- To access the shared data files, data consumers download encrypted data files of their interest from the cloud and then decrypt them.
- For the security purpose the data owner encrypts the data file's chunks and then store in the cloud.
- The data owner can change the policy over data files by updating the expiration time.
- The Data owner can have capable of manipulating the encrypted data file.
- The Cloud User who has a large amount of data to be stored in multiple clouds and have the permissions to access and manipulate stored data.
- The end user sends the request for corresponding file request and it will be processed in the cloud based on the queue and response to the end user.
- Admin can gives all access permissions as well as revoke permissions.
- For Inserting data, deleting data, modifying data admin can gives permission to data owner.

3.6 NON-FUNCTIONAL REQUIREMENTS

- **Performance:** The software shall support use of multiple users at a time. There are no other specific performance requirements that will affect development.

- **Portability:** Some of the attributes of software that relate to the ease of porting the software to other host machines and/or operating systems i.e, Java is used to develop the product. So it is easiest to port the software in any environment.

- **Maintainability**: The user will be able to reset all options and all stored user variables to default settings.

- **Reliability**: Data corruption is prevented by applying the possible backup procedures and techniques.

- **Usability:** A logical interface is essential to an easy to use system, speeding up common tasks.

- **Availability**: All cached data will be rebuilt during every startup. There is no recovery of user data if it is lost. Default values of system data will be assigned when necessary.

CHAPTER-4

SYSTEM SPECIFICATION

4. SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS

- Processor - Dual Core 1.6 GHz
- RAM - 2 GB
- HDD - 500 GB

4.2 SOFTWARE REQUIREMENTS

- Operating System - Windows 7 or above
- Programming Language - Python
- Tools - Google Collab or Jyupiter Notebook

CHAPTER-5
SYSTEM DESIGN

5. SYSTEM DESIGN

5.1 INTRODUCTION

The most creative and challenging phase of the life cycle is system and design. The term design describes a final system and the process by which it is developed. It refer to the technical specifications that will be applied in implementation the candidate system. The design may be defined as “the process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient details to permit its physical realization”.

The design’s goal is how the output is to be produced and in what format samples of the output and input are also presented. Second input data and database files have to be designed to meet the requirements of the proposed output. The processing phase is handled through the program construction and testing. Finally details related to justification of the system and an estimate of the impact of the candidate system on the users and the organization are documented and evaluated by management as a step toward implementation.

The importance of software design can be stated in a single word **“Quality”.** Design provides us with representation of software that can be assessed for quality. Design is the only way that we can accurately translate a customer’s requirements into a finished software product or system without design we risk building an unstable system, that might fail it small changes are made or may be difficult to test, or one who’s quality can’t be tested. So it is an essential phase in the development of a software product.

5.2 SDLC METHODOLOGY

The document ply vital role in the development of life cycle (SDLC) as it describes the complete requirement of the system. It means for use by developers and will be the basic during testing phase. Any changes made to the requirements in the future will have to go through formal change approval process.

SPIRAL MODEL was defined by Barry Boehm in his 1988 article, “A spiral model of Software Development and Enhancement.

.

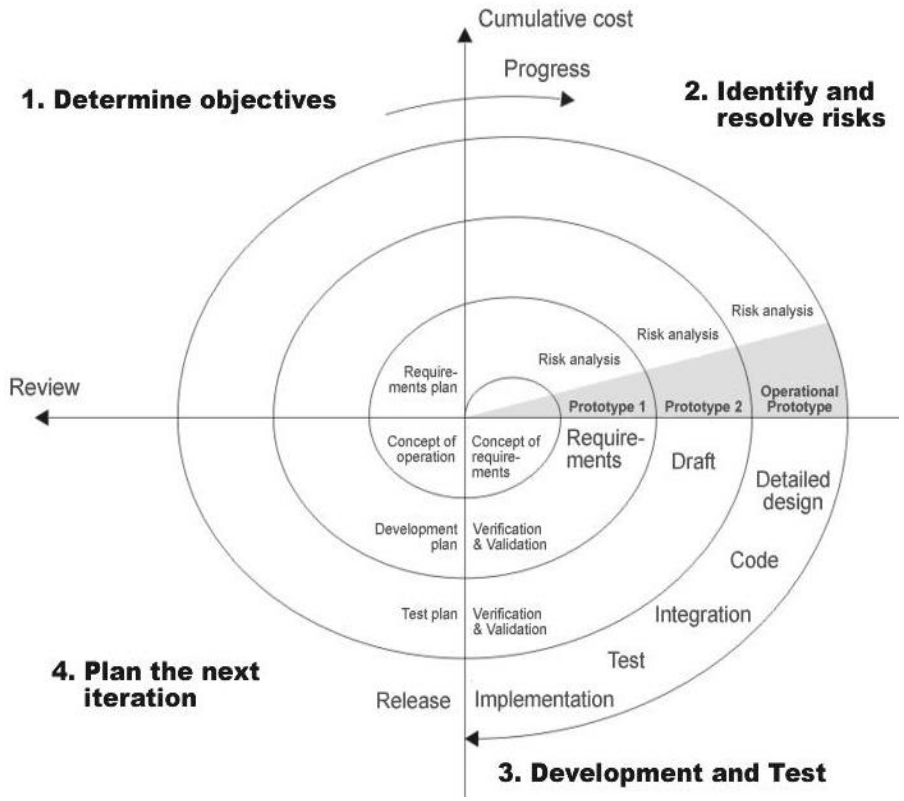


Fig 5.1: Stages in SDLC

STAGES IN SDLC

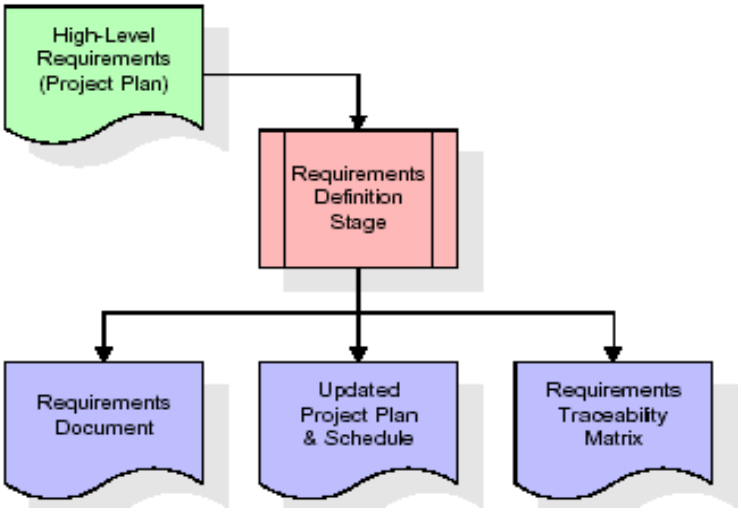
- Requirement Gathering
- Analysis
- Designing
- Coding
- Testing
- Maintenance

REQUIREMENTS GATHERING STAGE

The requirements gathering process takes as its input the goals identified in the high-level requirements section of the project plan.

Each goal will be refined into a set of one or more requirements. These requirements define the major functions of the intended application, define operational data areas and reference data areas, and define the initial data entities. Major functions include critical

processes to be managed, as well as mission critical inputs, outputs and reports. A user class hierarchy is developed and associated with these major functions, data areas, and data entities. Each of these definitions is termed a Requirement. Requirements are identified by unique requirement identifiers and, at minimum, contain a requirement title and textual description.



These requirements are fully described in the primary deliverables for this stage: the Requirements Document and the Requirements Traceability Matrix (RTM). The requirements document contains complete descriptions of each requirement, including diagrams and references to external documents as necessary. Note that detailed listings of database tables and fields are *not* included in the requirements document.

The title of each requirement is also placed into the first version of the RTM, along with the title of each goal from the project plan.

The purpose of the RTM is to show that the product components developed during each stage of the software development lifecycle are formally connected to the components developed in prior stages.

In the requirements stage, the RTM consists of a list of high-level requirements, or goals, by title, with a listing of associated requirements for each goal, listed by requirement title.

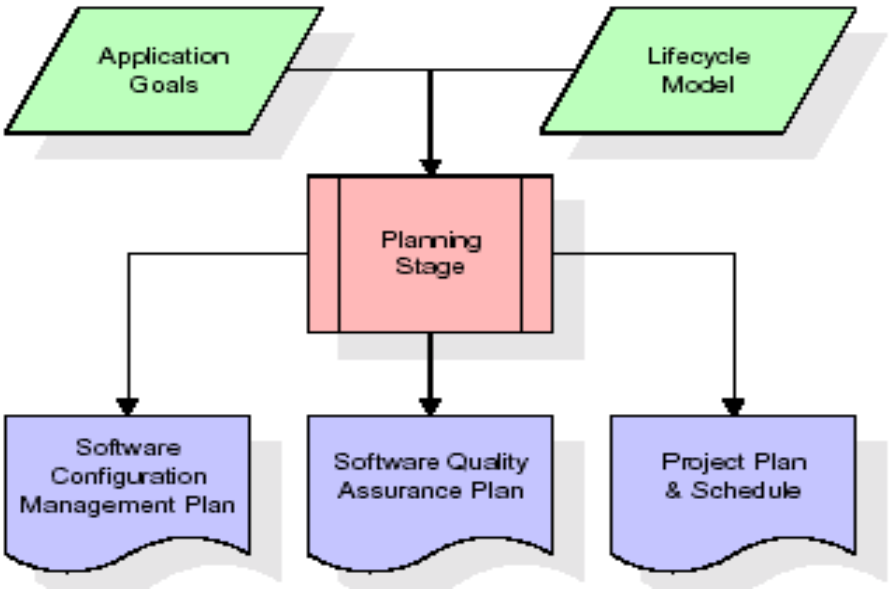
In this hierarchical listing, the RTM shows that each requirement developed during this stage is formally linked to a specific product goal.

The outputs of the requirements definition stage include the requirements document, the RTM, and an updated project plan.

- Feasibility study is all about identification of problems in a project.
- No. of staff required to handle a project is represented as Team Formation, in this case only modules are individual tasks will be assigned to employees who are working for that project.
- Project Specifications are all about representing of various possible inputs submitting to the server and corresponding outputs along with reports maintained by administrator

ANALYSIS STAGE:

The planning stage establishes a bird's eye view of the intended software product, and uses this to establish the basic project structure, evaluate feasibility and risks associated with the project, and describe appropriate management and technical approaches.



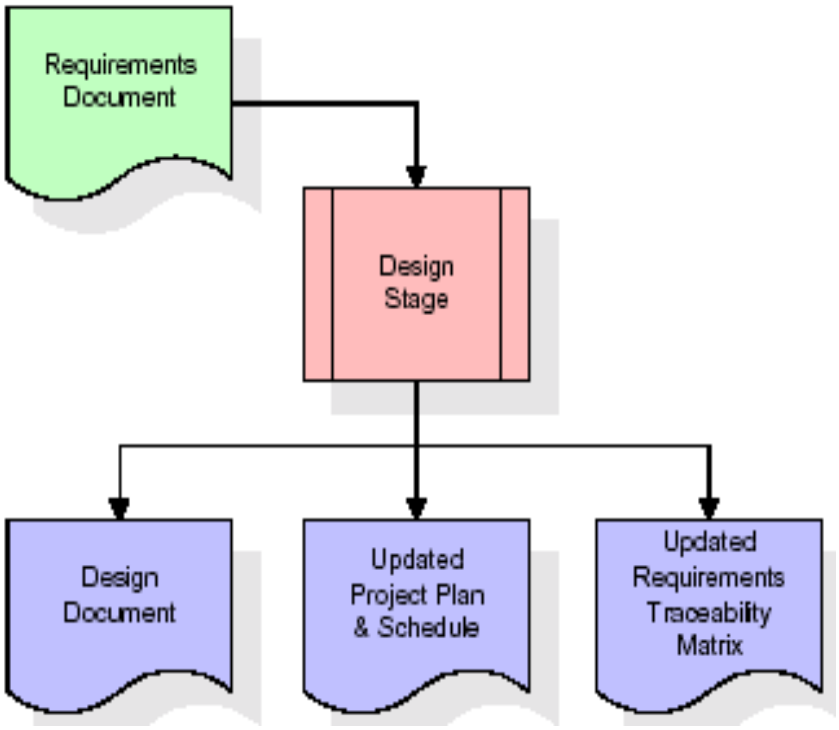
The most critical section of the project plan is a listing of high-level product requirements, also referred to as goals.

All of the software product requirements to be developed during the requirements definition stage flow from one or more of these goals.

The minimum information for each goal consists of a title and textual description, although additional information and references

to external documents may be included. The outputs of the project planning stage are the configuration management plan, the quality assurance plan,and the project plan and schedule, with a detailed listing of scheduled activities for the upcoming Requirements stage, and high level estimates of effort for the out stages.

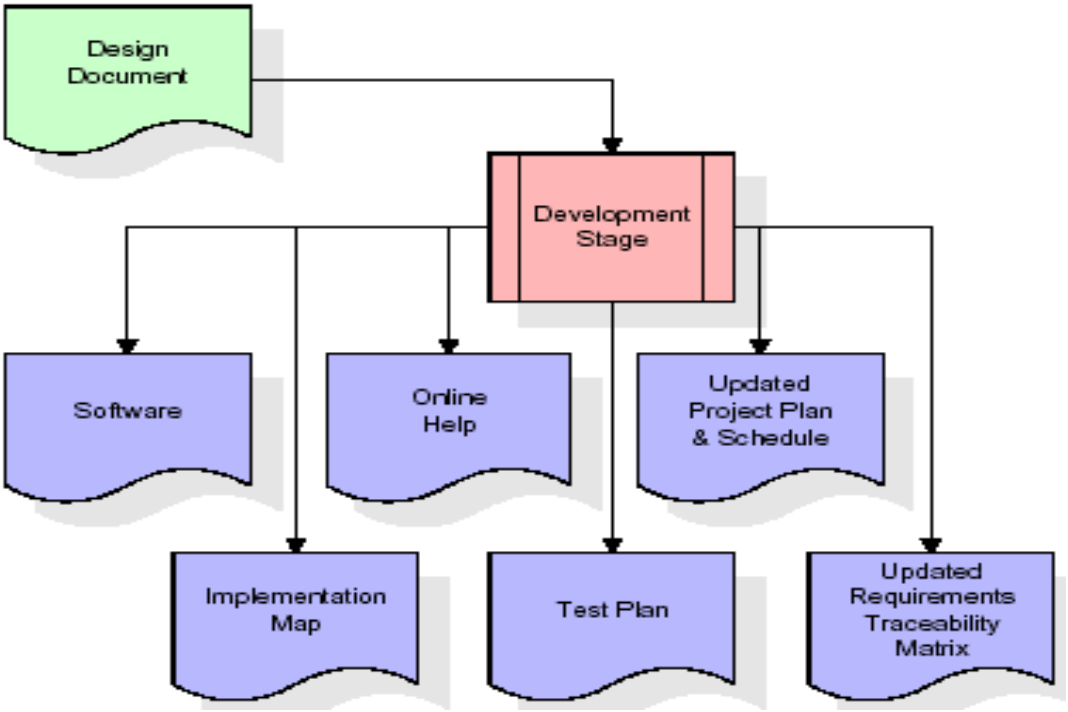
The design stage takes as its initial input the requirements identified in the approved requirements document. For each requirement, a set of one or more design elements will be produced as a result of interviews, workshops, and/or prototype efforts. Design elements describe the desired software features in detail, and generally include functional hierarchy diagrams, screen layout diagrams, tables of business rules, business process diagrams, pseudo code, and a complete entity-relationship diagram with a full data dictionary. These design elements are intended to describe the software in sufficient detail that skilled programmers may develop the software with minimal additional input.



When the design document is finalized and accepted, the RTM is updated to show that each design element is formally associated with a specific requirement.

The outputs of the design stage are the design document, an updated RTM, and an updated project plan.

The development stage takes as its primary input the design elements described in the approved design document. For each design element, a set of one or more software artifacts will be produced. Software artifacts include but are not limited to menus, dialogs, data management forms, data reporting formats, and specialized procedures and functions. Appropriate test cases will be developed for each set of functionally related software artifacts, and an online help system will be developed to guide users in their interactions with the software.



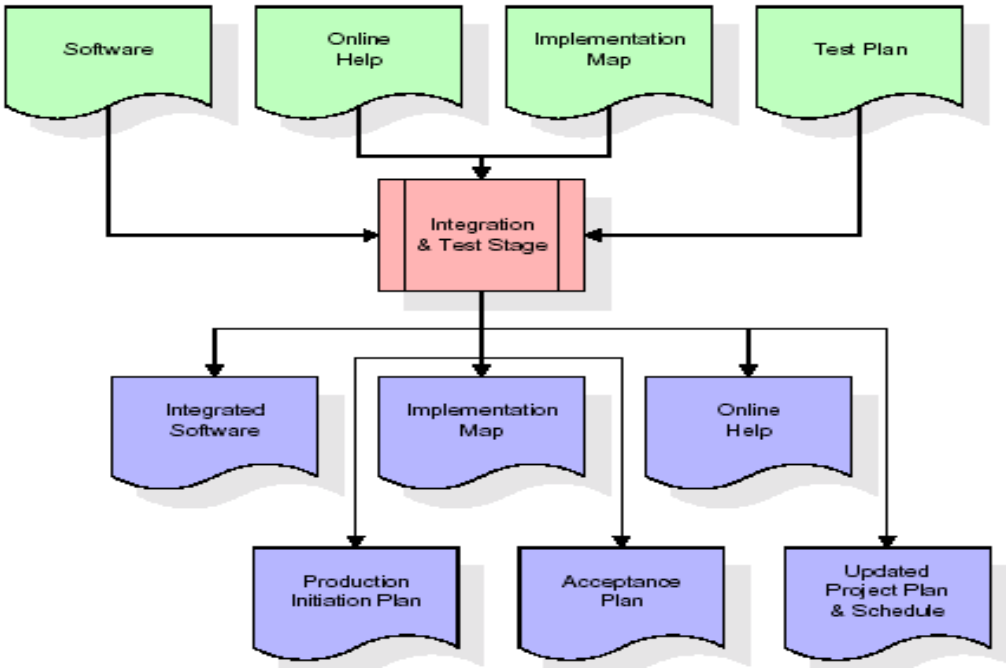
The RTM will be updated to show that each developed artifact is linked to a specific design element, and that each developed artifact has one or more corresponding test case items. At this point, the RTM is in its final configuration. The outputs of the development stage include a fully functional set of software that satisfies the requirements and design elements previously documented, an online help system that describes the operation of the software.

An implementation map that identifies the primary code entry points for all major system functions, a test plan that describes the test cases to be used to validate the correctness and completeness of the software, an updated RTM, and an updated project plan.

INTEGRATION & TEST STAGE

During the integration and test stage, the software artifacts, online help, and test data are migrated from the development environment to a separate test environment. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite confirms a robust and complete migration capability. During this stage, reference data is finalized for production use and production users are identified and linked to their appropriate roles.

The final reference data (or links to reference data source files) and production user list are compiled into the Production Initiation Plan.

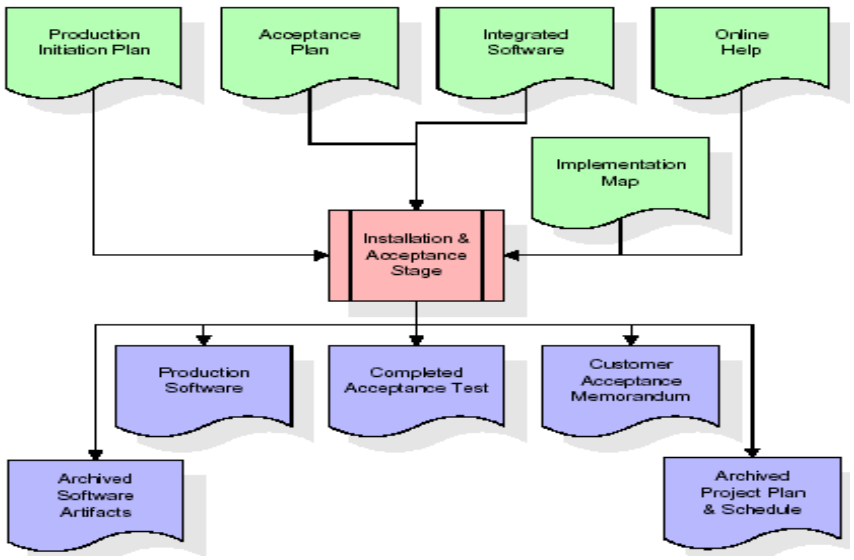


The outputs of the integration and test stage include an integrated set of software, an online help system, an implementation map, a production initiation plan that describes reference data and production users, an acceptance plan which contains the final suite of test cases, and an updated project plan.

INSTALLATION & ACCEPTANCE TEST

During the installation and acceptance stage, the software artifacts, online help, and initial production data are loaded onto the production server. At this point, all test cases are run to verify the correctness and completeness of the software. Successful execution of the test suite is a prerequisite to acceptance of the software by the customer.

After customer personnel have verified that the initial production data load is correct and the test suite has been executed with satisfactory results, the customer formally accepts the delivery of the software.



The primary outputs of the installation and acceptance stage include a production application, a completed acceptance test suite, and a memorandum of customer acceptance of the software. Finally, the PDR enters the last of the actual labour data into the project schedule and locks the project as a permanent project record. At this point the PDR "locks" the project by archiving all software items, the implementation map, the source code, and the documentation for future reference.

MAINTENANCE

Outer rectangle represents maintenance of a project, Maintenance team will start with requirement study, understanding of documentation later employees will be assigned work and they will undergo training on that particular assigned category.

5.3 INPUT DESIGN

Input design includes data mediums used for inputting data and validations that are to be done during data entry. Different messages regarding data are given to guide users during data entry. Validation checks are done for each input.

Data entry screens are designed so that the system interacts with the user in providing an effective dialogue. Fields in the screen are logically arranged to help the user.

The design is the process of converting the user-originated inputs into a computer-based format. The goal of the input design is to make the data entry easier, logical and free from error. Errors in the input data are controlled by input design.

The application has been developed in a user-friendly manner. The windows have been designed in such a way that during the processing the cursor is placed in the position where the data must be entered. If any of the data going into the system is wrong then the process and output will magnify these errors.

The decisions made during design of input are:

1) To achieve the highest possible level of accuracy.

2) To provide a list of possible choices and help while accepting the input for an important field wherever possible outputs from computer system are required primarily to communicate the results of processing to the users. They are also used to provide a permanent copy of these results for later consultation/verification.

5.4 OUTPUT DESIGN

Output refers to the results and information that are generated by the system. Output is the main reason for developing the system and based on this, the usefulness and applicability of system are evaluated.

Outputs from computer systems are required primarily to communicate the results of processing to users. Efficiently designed outputs enhance the understandability of the information.

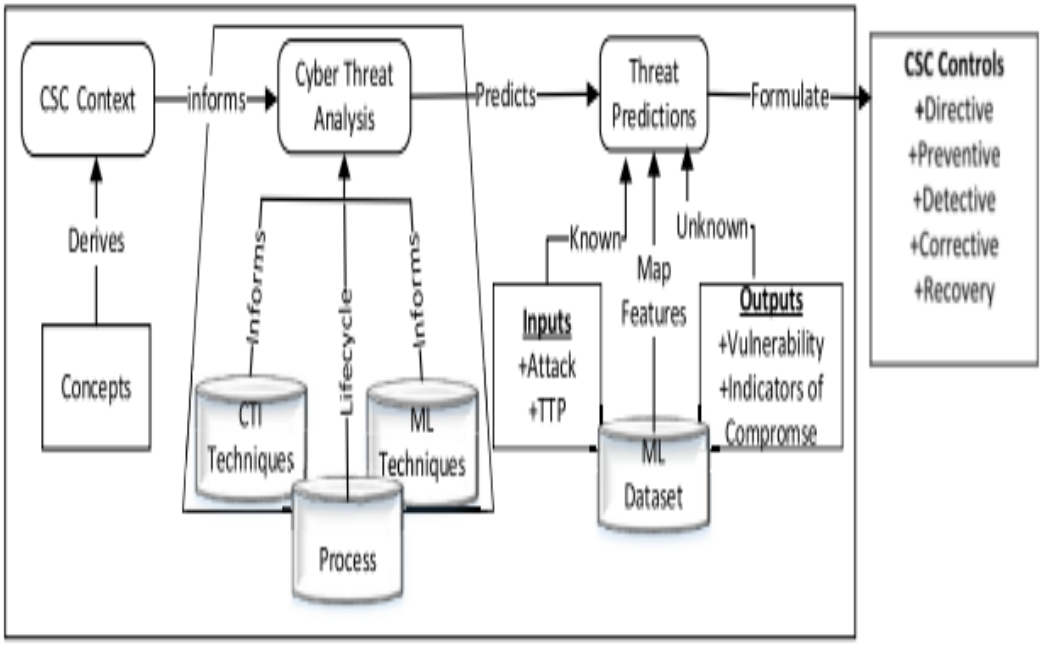
According to the requirements of the system, various types of outputs are considered and designed as follows.

Internal outputs, whose destination is within the organization and which require careful design because they are the user's main interface with the computer.

Interactive outputs, in which the user communication with the Computer is essential.

5.5 SYSTEM ARCHITECTURE

ARCHITECTURE DIAGRAM



5.6 DATA FLOW DIAGRAM

The overall logical structure of a database can be expressed graphically by an **E-R diagram.** The relative simplicity and pictorial clarity of this diagramming technique may well account in large part for the widespread use of the E-R model. Such a diagram consists of the following major components.

Rectangles: Represent Entity Sets.

Ellipses: Represent attributes.

Diamonds: Represent relationship sets

Lines: Link attributes to entity sets and entity sets

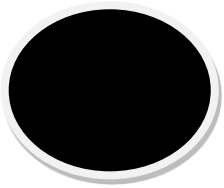
A graphical tool used to describe and analyze the moment of data through a system manual or automated including the process, stores of data, and delays in the system.

Data Flow Diagrams are the central tool and the basis from which other components are developed. The transformation of data from input to output, through processes, may be described logically and independently of the physical components associated with the system. The DFD is also know asa data flow graph or a bubble chart.

DFDs are the model of the proposed system. They clearly should show the requirements on which the new system should be built. Later during design activity this is taken as the basis for drawing the system's structure charts. The Basic Notation used to create a DFD's are as follows:

1. **Dataflow:** Data move in a specific direction from an origin to a destination.

2. **Process:** People, procedures, or devices that use or produce (Transform) Data. The physical component is not identified.



3. **Source:** External sources or destination of data, which may be People, programs, organizations or other entities.

4. **Data Store:** Here data are stored or referenced by a process in the System.

5.7 UML DIAGRAM

Unified Modelling Language is a language available to perform modelling of software. A model is simplification of reality. A model provides the blue print of the system, model encompasses detailed plans.

BUILDING BLOCKS OF THE UML

The vocabulary of the UML encompasses three kinds of building blocks.

1. Things.
2. Relationships.
3. Diagrams.

Things in the UML

Things are the abstractions that are first-class citizen in a model.

There are four kinds of things in the UML.

1. Structure things.
2. Behavioural things.
3. Grouping things.
4. Annotational things.

These things are the basic object-oriented building blocks of the UML. You use them to write well-formed models.

RELATIONSHIPS IN THE UML

Things can be connected to logically are physically with the help of relationship in object oriented modelling. These are four kinds of relationships in the UML.

1. Dependency.
2. Association.
3. Generalization.
4. Realization.

DIAGRAMS IN THE UML

A diagram is a graphical representation of a set of elements. These are nine kinds of diagrams in the UML.

1. Class diagram.
2. Object diagram
3. Use Case diagram.
4. Sequence diagram.
5. Collaboration diagram
6. Activity diagram.
7. Component diagram
8. State chart diagram.
9. Deployment diagram.

5.7.1 CLASS DIAGRAM:

Class diagrams area unit the foremost common diagrams employed in UML. Category diagram consists of categories, interfaces, associations and collaboration. Category diagrams primarily represent the thing directed read of a system that is static in nature. Active category is employed in a very category diagram to represent the concurrency of the system.

Class diagram represents the thing orientation of a system. Therefore it's usually used for development purpose. This can be the foremost wide used diagram at the time of system construction.

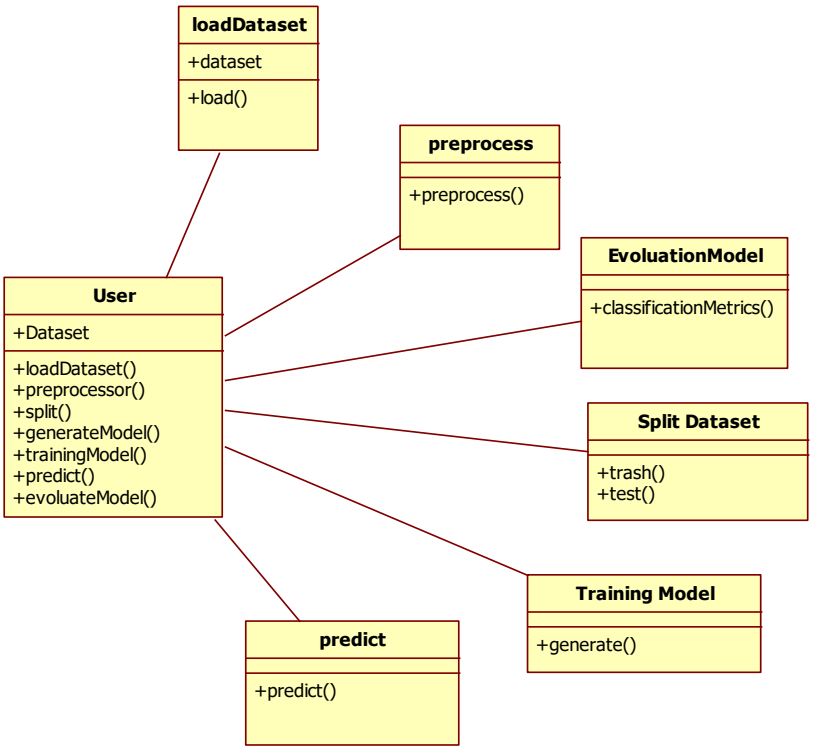


Fig 5.7.2 Class Diagram

5.7.2 USECASE DIAGRAM

Use case diagram shows a set of use cases and actors (a special kind of class) and their relationship. Use case diagrams address the static use case view of a system.

These diagrams are especially important in organizing and modelling the behavioural of a system both sequence and collaboration diagrams are kind of interaction diagram.

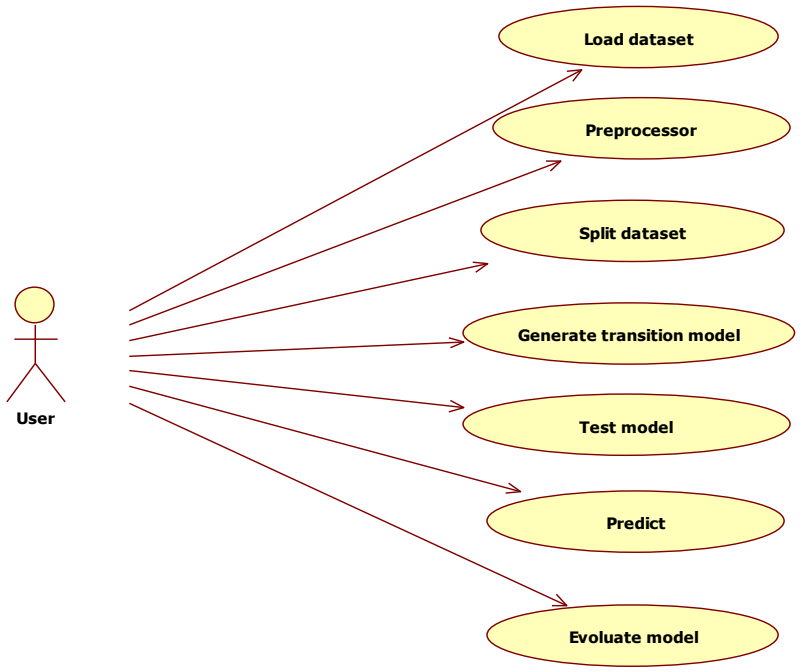


Fig 5.7.1 Use Case Diagram

5.7.3 SEQUENCE DIAGRAM

A sequence diagram in Unified Modelling 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.

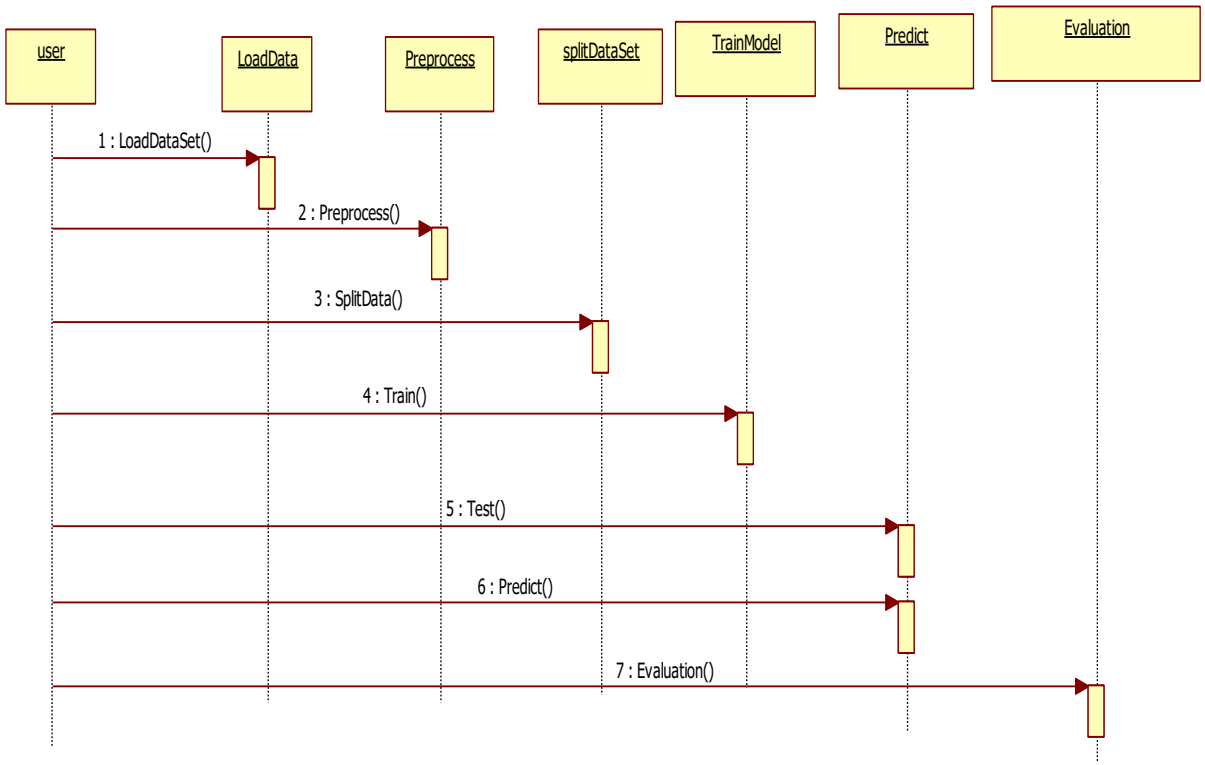


Fig 5.7.3 Sequence Diagram

5.7.4 ACTIVITY DIAGRAM

An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed. We can depict both sequential processing and concurrent processing of activities using an activity diagram. They are used in business and process modelling where their primary use is to depict the dynamic aspects of a system. An activity diagram is very similar to a flowchart.

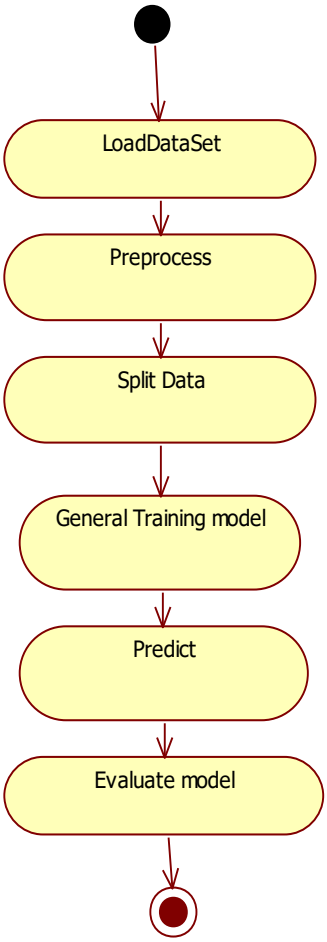


Fig. 5.7.4: Activity Diagram

CHAPTER-6

IMPLEMENTATION

6.IMPLEMENTATION

6.1 MODULES DESCRIPTION

DATA COLLECTION

In this module, we collect the malware detection dataset from kaggle.com. In the dataset we have 162263 records and 57 features are available. This data can be used to train and test the applied models performance.

PRE-PROCESSING

In this module, we processing the data in which we find the any missing values, check data type of attributes or features that provide information about the data has either continuous data or binary data. If any continuous data, we need to convert the continuous data into binary data. After converting data, we perform data normalization process for better understanding of data.

SPLIT DATASET

In this module, after processing the data set we split the data as a training and testing data. The training data is used to train the models to predict malicious data and Test data can be used for check the efficiency of the models.

PREDICTION

In this module, we apply different ML algorithms such as Logistic Regression, Decision Tree, Naive Bayes and Random Forest. Using these models, we predict the malware in the dataset. Calculate the prediction accuracies of the applied models for perform comparative analysis.

EVALUATION

In this module, we construct and calculate we confusion matrix and classification metrics to further evaluate the models.

6.2 EXPERIMENTAL ANALYSIS

6.2.1 INTRODUCTION

This section presents and analyses the results of the threat prediction. We follow a number of assessment parameters such as attack probability, TTP, vulnerable spots, and IoC for this purpose. The attack probability figures are derived from Table 2. The propagation is determined using a probability scale of 0–100%. A percentage score was given after calculating the degree of severity of each manipulation. Form low (≤15%), medium (16% to 59%), or high (above 60%).

6.2.2 PREDICTION OF AN ATTACK PROBABILITY.

Table 3 presents the performance of the classifications of LR, DT, SVM, RF algorithms in identifying the various responses of cyberattacks based on the given malicious attack. From the table, LR achieved an accuracy of 66%, DT, 63% SVM 62% and RF 66%. Comparing the performance of the classifiers, LR and RF both performed better for the Precision, Recall and F-Score, whilst DT and SVM received a low precision, recall and F-score. Comparing that to the attack's categories signifies that Malware, Ransomware and spyware attacks identified different types of responses with 85% accuracy.

6.2.3 PREDICTION OF TTP DEPLOYED BASED ON THE RESPONSE OF THE CYBER ATTACK

Table 4 presents the performance of the classification algorithms in identifying the various TTPs deployed, and responses based on the given attack vectors. Comparing the TTPs against the attack categories, XSS, session hijacking and RAT attack, DT and SVM achieved a low content for the low precision recall and F-score. However, LR received the highest precision and F-score for malware attack with 83% accuracy for TTPs deployed. Furthermore, ransomware and spyware attacks identified different types of responses for the TTPs with 83% accuracy for the harmonic mean in identifying the attack vectors being rootkit, email attachments and RAT

6.2.4 PREDICTION OF VULNERABLE SPOTS BASED ON THE DIFFERENT TYPES OF RESPONSES OF CYBER ATTACKS

Table 5 presents the performance of the various classifications of the LR, DT, SVM and RF algorithms in identifying the vulnerable spots based on the different types of responses of cyberattacks. The vulnerable spots were identified from the CSC system probable threats table in [2] and used the manipulations figures for precision, recall and F-Score. LR and RF achieved a similar accuracy of 87% for the precision and F-score the successful attacks that signify the probability of exploits on the network nodes. Further, attacks such as malware and ransomware received higher precision based on the exploits and TTPS deployed with 92% accuracy. Whilst spear phishing, session hijacking and DDoS performs lower with the DT and SVM classifiers.

6.2.5 PREDICATION OF INDICATORS OF COMPROMISE (IOC).

Table 6 presents the performance variations of the various classifications algorithms that identify what constitutes as indicators of compromise. With DDoS attack, RF presented the highest precision values of 83% compare to SVM indicating the extent of compromises on the network.

TABLE 6.1: Predict the probability of an attack from the various endpoints.

ALGORITHMS	R			DT			SVM			RF		
ACCURACY (%)	66			63			62			66		
ATTACKS	P	R	F	P	R	F	P	R	F	P	R	F
XSS/Session Hijacking	0.88	0.38	0.65	0.58	0.42	0.68	0.55	0.38	0.63	0.88	0.38	0.65
Spyware/Ransomware	0.90	0.55	0.75	0.85	0.37	0.70	0.65	0.45	0.63	0.90	0.55	0.75
Spear Phishing	0.81	0.17	0.71	0.55	0.28	0.66	0.58	0.36	0.63	0.81	0.17	0.71
Session Hijacking	0.73	0.36	0.62	0.48	0.35	0.61	0.55	0.38	0.63	0.73	0.36	0.62
Rootkit/DDoS	0.56	0.37	0.65	0.57	0.33	0.58	0.53	0.35	0.63	0.56	0.37	0.65
RAT/Island Hopping	0.68	0.30	0.73	0.55	0.22	0.69	0.51	0.25	0.63	0.68	0.30	0.73
Ransomware/Malware	0.88	0.53	0.60	0.59	0.26	0.71	0.54	0.31	0.63	0.88	0.53	0.60
Malware/Spyware	0.81	0.48	0.68	0.58	0.51	0.73	0.55	0.45	0.63	0.81	0.48	0.68
DDoS	0.78	0.36	0.65	0.55	0.33	0.55	0.51	0.32	0.53	0.78	0.36	0.65

TABLE 6.2: Identify the different TTP deployed based on the response of cyberattacks

ALGORITHMS	LR			DT			SVM			RF		
ACCURACY (%)	66			63			62			66		
ATTACKS	P	R	F	P	R	F	P	R	F	P	R	F
XSS/Session Hijacking	0.82	0.26	0.55	0.55	0.31	0.61	0.55	0.27	0.56	0.82	0.26	0.55
Spyware/Ransomware	0.88	0.51	0.71	0.65	0.33	0.62	0.65	0.31	0.61	0.88	0.51	0.71
Spear Phishing	0.71	0.23	0.61	0.53	0.22	0.56	0.58	0.36	0.59	0.71	0.23	0.61
Session Hijacking	0.63	0.26	0.58	0.52	0.28	0.52	0.56	0.38	0.48	0.63	0.26	0.58
Rootkit/DDoS	0.51	0.27	0.63	0.51	0.31	0.58	0.48	0.35	0.57	0.51	0.27	0.63
RAT/Island Hopping	0.68	0.28	0.68	0.54	0.21	0.61	0.51	0.25	0.58	0.68	0.28	0.68
Ransomware/Malware	0.86	0.44	0.66	0.58	0.22	0.65	0.59	0.31	0.62	0.86	0.44	0.66
Malware/Spyware	0.79	0.41	0.67	0.65	0.51	0.63	0.55	0.45	0.61	0.79	0.41	0.67
DDoS	0.71	0.36	0.61	0.55	0.33	0.55	1.55	0.32	0.53	0.71	0.36	0.61

TABLE 6.3: Predict vulnerable spots based on the different types of cyberattacks.

ALGORITHMS	LR			DT			SVM			RF		
ACCURACY (%)	66			63			62			66		
ATTACKS	P	R	F	P	R	F	P	R	F	P	R	F
XSS/Session Hijacking	0.63	0.60	0.61	0.65	0.61	0.62	0.61	0.59	0.60	0.62	0.59	0.61
Spyware/Ransomware	0.85	0.83	0.80	0.86	0.81	0.83	0.82	0.79	0.81	0.83	0.78	0.80
Spear Phishing	0.68	0.62	0.66	0.63	0.59	0.61	0.64	0.60	0.62	0.63	0.61	0.68
Session Hijacking	0.66	0.61	0.64	0.65	0.61	0.64	0.62	0.59	0.60	0.63	0.60	0.62
Rootkit/DDoS	0.64	0.60	0.61	0.63	0.61	0.58	0.61	0.57	0.59	0.64	0.38	0.58
RAT/Island Hopping	0.64	0.61	0.63	0.65	0.62	0.64	0.64	0.61	0.62	0.64	0.33	0.58
Ransomware/Malware	0.84	0.81	0.82	0.85	0.81	0.84	0.61	0.58	0.60	0.75	0.55	0.62
Malware/Spyware	0.82	0.77	0.81	0.86	0.83	0.85	0.85	0.81	0.83	0.66	0.45	0.69
DDoS	0.65	0.61	0.62	0.64	0.60	0.63	0.62	0.59	0.61	0.75	0.33	0.62

TABLE 6.4: Indicators of compromise (IOC)

ALGORITHMS	LR			DT			SVM			RF		
ACCURACY (%)	66			63			62			66		
ATTACKS	P	R	F	P	R	F	P	R	F	P	R	F
XSS/Session Hijacking	0.68	0.63	0.66	0.55	0.42	0.61	0.51	0.38	0.63	0.68	0.37	0.71
Spyware/Ransomware	0.80	0.8	0.75	0.85	0.55	0.70	0.65	0.45	0.63	0.78	0.52	0.76
Spear Phishing	0.81	0.17	0.71	0.55	0.65	0.70	0.55	0.45	0.63	0.77	0.17	0.68
Session Hijacking	0.73	0.66	0.62	0.55	0.65	0.70	0.55	0.45	0.63	0.73	0.65	0.62
Rootkit/DDoS	0.56	0.37	0.60	0.55	0.65	0.70	0.55	0.45	0.63	0.56	0.37	0.59
RAT/Island Hopping	0.68	0.30	0.33	0.55	0.65	0.70	0.55	0.45	0.63	0.68	0.30	0.63
Ransomware/Malware	0.70	0.33	0.62	0.55	0.65	0.70	0.55	0.45	0.63	0.72	0.33	0.60
Malware/Spyware	0.74	0.48	0.65	0.55	0.65	0.70	0.55	0.45	0.63	0.71	0.48	0.65
DDoS	0.68	0.56	0.65	0.55	0.65	0.70	1.55	0.45	0.63	0.68	0.56	0.57

LR received the highest precision and F-score for malware and spyware attacks, whereas RF and LR received the similar precision, recall and F-score.

CHAPTER-7
TECHNOLOGY DESCRIPTION

7. TECHNOLOGY DESCRIPTION

INTRODUCTION TO PYTHON

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system capable of collecting reference cycles. Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3. The Python 2 language, i.e. Python 2.7.x, was officially discontinued on 1 January 2020 (first planned for 2015) after which security patches and other improvements will not be released for it.[32][33] With Python 2's end-of-life, only Python 3.5.x and later are supported. Python interpreters are available for many operating systems. A global community of programmers develops and maintains Python, an open source [35] reference implementation. A non-profit organization, the Python Software Foundation, manages and directs resources for Python and Python development.

SYNTAX AND SEMANTICS

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation.

Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

INDENTATION

Main article: Python syntax and semantics § Indentation

Python uses whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. Thus, the program's visual structure accurately represents the program's semantic structure.

This feature is sometimes termed the off-side rule, which some other languages share, but in most languages, indentation doesn't have any semantic meaning.

STATEMENTS AND CONTROL FLOW

Python's statements include (among others):

The assignment statement (token '=', the equals sign). This operates differently than in traditional imperative programming languages, and this fundamental mechanism (including the nature of Python's version of variables) illuminates many other features of the language. Assignment in C, e.g., x = 2, translates to "typed variable name x receives a copy of numeric value 2". The (right-hand) value is copied into an allocated storage location for which the (left-hand) variable name is the symbolic address. The memory allocated to the variable is large enough (potentially quite large) for the declared type. In the simplest case of Python assignment, using the same example, x = 2, translates to "(generic) name x receives a reference to a separate, dynamically allocated object of numeric (int) type of value 2." This is termed binding the name to the object. Since the name's storage location doesn't contain the indicated value, it is improper to call it a variable. Names may be subsequently rebound at any time to objects of greatly varying types, including strings, procedures, complex objects with data and methods, etc. Successive assignments of a common value to multiple names, e.g., x = 2; y = 2; z = 2 result in allocating storage to (at most) three names and one numeric object, to which all three names are bound.

Since a name is a generic reference holder it is unreasonable to associate a fixed data type with it. However at a given time a name will be bound to some object, which will have a type; thus there is dynamic typing.

- The if statement, which conditionally executes a block of code, along with else and elf (a contraction of else-if).
- The for statement, which iterates over an inerrable object, capturing each element to a local variable for use by the attached block.
- The while statement, which executes a block of code as long as its condition is true.
- The try statement, which allows exceptions raised in its attached code block to be caught and handled by except clauses; it also ensures that clean-up code in a finally block will always be run regardless of how the block exits.
- The raise statement, used to raise a specified exception or re-raise a caught exception.
- The class statement, which executes a block of code and attaches its local namespace to a class, for use in object-oriented programming.
- The def statement, which defines a function or method.
- The with statement, from Python 2.5 released in September 2006, which encloses a code block within a context manager (for example, acquiring a lock before the block of code is run and releasing the lock afterwards, or opening a file and then closing it), allowing Resource Acquisition Is Initialization (RAII)-like behaviour and replaces a common try/finally idiom.
- The break statement, exits from the loop.
- The continue statement, skips this iteration and continues with the next item.
- The pass statement, which serves as a NOP. It is syntactically needed to create an empty code block.
- The assert statement, used during debugging to check for conditions that ought to apply.
- The yield statement, which returns a value from a generator function. From Python 2.5, yield is also an operator. This form is used to implement coroutines.

The import statement, which is used to import modules whose functions or variables can be used in the current program. There are three ways of using import: import <module name> [as <alias>] or from <module name> import * or from <module name> import <definition 1> [as <alias 1>], <definition 2> [as <alias 2>],

The print statement was changed to the print () function in Python 3.

Python does not support tail call optimization or first-class continuations, and, according to Guido van Rossum, it never will. However, better support for coroutine-like functionality is provided in 2.5, by extending Python's generators. Before 2.5, generators were lazy iterators; information was passed unidirectionally out of the generator. From Python 2.5, it is possible to pass information back into a generator function, and from Python 3.3, the information can be passed through multiple stack levels.

EXPRESSIONS

Some Python expressions are similar to languages such as C and Java, while some are not:

Addition, subtraction, and multiplication are the same, but the behavior of division differs. There are two types of divisions in Python. They are floor division (or integer division) // and floating point/division. Python also added the ** operator for exponentiation.

From Python 3.5, the new @ infix operator was introduced. It is intended to be used by libraries such as NumPy for matrix multiplication.

From Python 3.8, the syntax :=, called the 'walrus operator' was introduced. It assigns values to variables as part of a larger expression.

In Python, == compares by value, versus Java, which compares numeric by valueand objects by reference. (Value comparisons in Java on objects can be performed with the equals() method.) Python's is operator may be used to compare object identities (comparison by reference). In Python, comparisons may be chained, for example a <= b <= c.

Python uses the words and, or, not for its Boolean operators rather than the symbolic &&, ||, ! used in Java and C.

Python has a type of expression termed a list comprehension. Python 2.4 extended list comprehensions into a more general expression termed a generator expression.

Anonymous functions are implemented using lambda expressions; however, these are limited in that the body can only be one expression.

Conditional expressions in Python are written as x if c else y (different in order of operands from the c?x: y operator common to many other languages).

Python makes a distinction between lists and tuples. Lists are written as [1, 2, 3], are mutable, and cannot be used as the keys of dictionaries (dictionary keys must be immutable in Python). Tuples are written as (1, 2, 3), are immutable and thus can be used as the keys of dictionaries, provided all elements of the tuple are immutable. The + operator can be used to concatenate two tuples, which does not directly modify their contents, but rather produces a new tuple containing the elements of both provided tuples. Thus, given the variable t initially equal to (1, 2, 3), executing t = t + (4, 5) first evaluates t + (4, 5), which yields (1, 2, 3, 4, 5), which is then assigned back to t, thereby effectively "modifying the contents" of t, while conforming to the immutable nature of tuple objects. Parentheses are optional for tuples in unambiguous contexts.

Python features sequence unpacking wherein multiple expressions, each evaluating to anything that can be assigned to (a variable, a writable property, etc.), are associated in the identical manner to that forming tuple literals and, as a whole, are put on the left hand side of the equal sign in an assignment statement. The statement expects an inerrable object on the right-hand side of the equal sign that produces the same number of values as the provided writable expressions when iterated through, and will iterate through it, assigning each of the produced values to the corresponding expression on the left.

Python has a "string format" operator %. This function analogous to printf format strings in C, e.g. "spam=%s eggs=%d" % ("blah", 2) evaluates to "spam=blah eggs=2".

In Python 3 and 2.6+, this was supplemented by the format() method of the str class, e.g. "spam={0} eggs={1}".format("blah", 2). Python 3.6 added "f-strings": blah = "blah"; eggs = 2; f'spam={blah} eggs={eggs}'.

PYTHON HAS VARIOUS KINDS OF STRING LITERALS:

Strings delimited by single or double quote marks. Unlike in Unix shells, Perl and Perl-influenced languages, single quote marks and double quote marks function identically. Both kinds of string use the backslash (\) as an escape character. String interpolation became available in Python 3.6 as "formatted string literals".

Triple-quoted strings, which begin and end with a series of three single or double quote marks. They may span multiple lines and function like here documents in shells, Perl and Ruby.

Raw string varieties, denoted by prefixing the string literal with an r. Escape sequences are not interpreted; hence raw strings are useful where literal backslashes are common, such as regular expressions and Windows-style paths. Compare "@-quoting" in C#.

Python has array index and array slicing expressions on lists, denoted as a[key], a[start:stop] or a[start:stop:step]. Indexes are zero-based, and negative indexes are relative to the end. Slices take elements from the start index up to, but not including, the stop index.

The third slice parameter, called step or stride, allows elements to be skipped and reversed. Slice indexes may be omitted, for example a[:] returns a copy of the entire list. Each element of a slice is a shallow copy.

In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp, Scheme, or Ruby. This leads to duplicating some functionality. For example:

List comprehensions vs. for-loops

Conditional expressions vs. if blocks

The eval() vs. exec() built-in functions (in Python 2, exec is a statement); the former is for expressions, the latter is for statements.

Statements cannot be a part of an expression, so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements.

A particular case of this is that an assignment statement such as a = 1 cannot form part of the conditional expression of a conditional statement.

This has the advantage of avoiding a classic C error of mistaking an assignment operator = for an equality operator == in conditions: if (c = 1) { ... } is syntactically valid (but probably unintended) C code but if c = 1: ... causes a syntax error in Python.

METHODS

Methods on objects are functions attached to the object's class; the syntax instance. Method(argument) is, for normal methods and functions, syntactic sugar for Class Method(instance, argument). Python methods have an explicit self-parameter to access instance data, in contrast to the implicit self (or this) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby).

APPLICATIONS OF PYTHON

As mentioned before, Python is one of the most widely used language over the web. I'm going to list few of them here:

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-maintaining.

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.

INSTALLATION STEPS OF PYTHON

Installing and using Python on Windows 10 is very simple. The installation procedure involves just three steps:

- Download the binaries
- Run the Executable installer
- Add Python to PATH environmental variables

To install Python, you need to download the official Python executable installer. Next, you need to run this installer and complete the installation steps. Finally, you can configure the PATH variable to use python from the command line.

Step 1: Download the Python Installer binaries

- Open the official Python website in your web browser. Navigate to the Downloads tab for Windows.
- Choose the latest Python 3 release. In our example, we choose the latest Python 3.7.3 version. Click on the link to download Windows x86 executable installer if you are using a 32-bit installer.
- In case your Windows installation is a 64-bit system, then download Windows x86-64 executable installer.

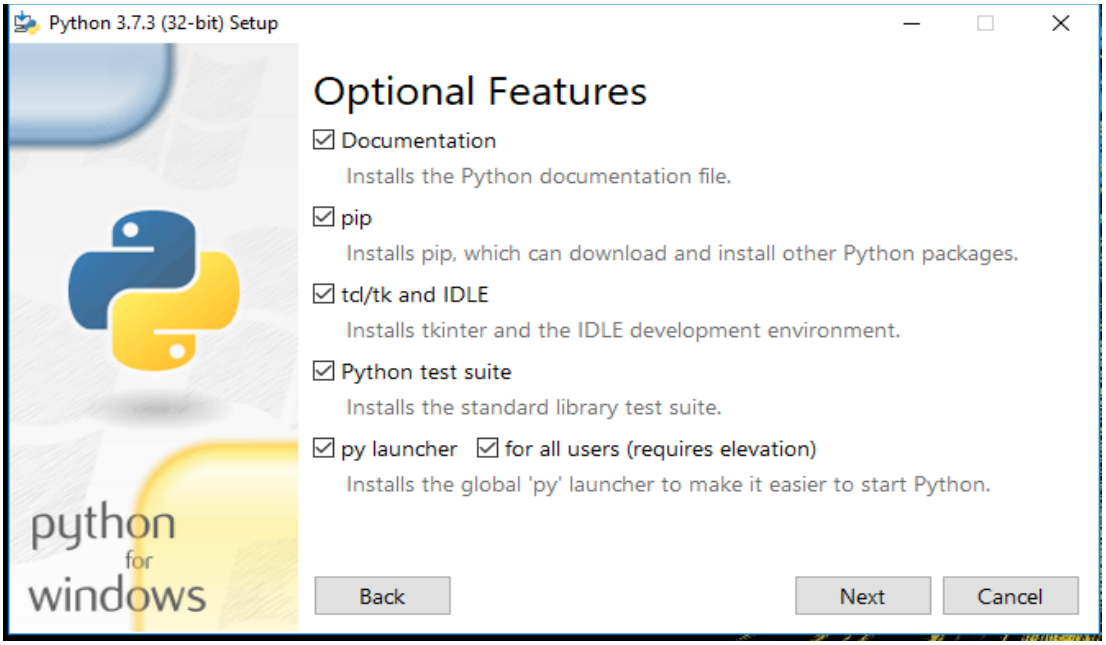
Step 2: Run the Executable Installer

1. Once the installer is downloaded, run the Python installer.
2. Check the Install launcher for all users check box. Further, you may check the Add Python 3.7 to path check box to include the interpreter in the execution path.

3. Select **Customize installation**.

Choose the optional features by checking the following check boxes:

1. Documentation
2. pip
3. tcl/tk and IDLE (to install tkinter and IDLE)
4. Python test suite (to install the standard library test suite of Python)
5. Install the global launcher for `.py` files. This makes it easier to start Python
6. Install for all users.

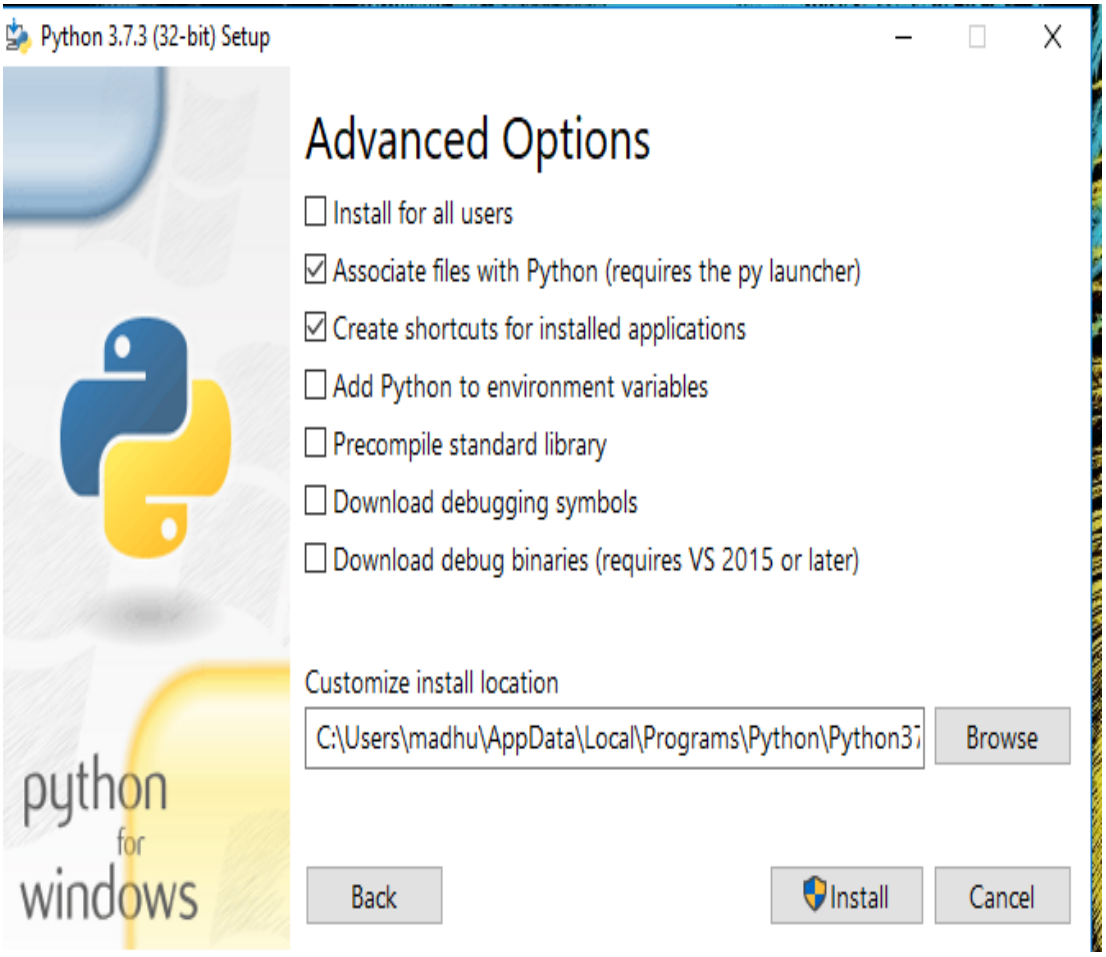


Click Next.

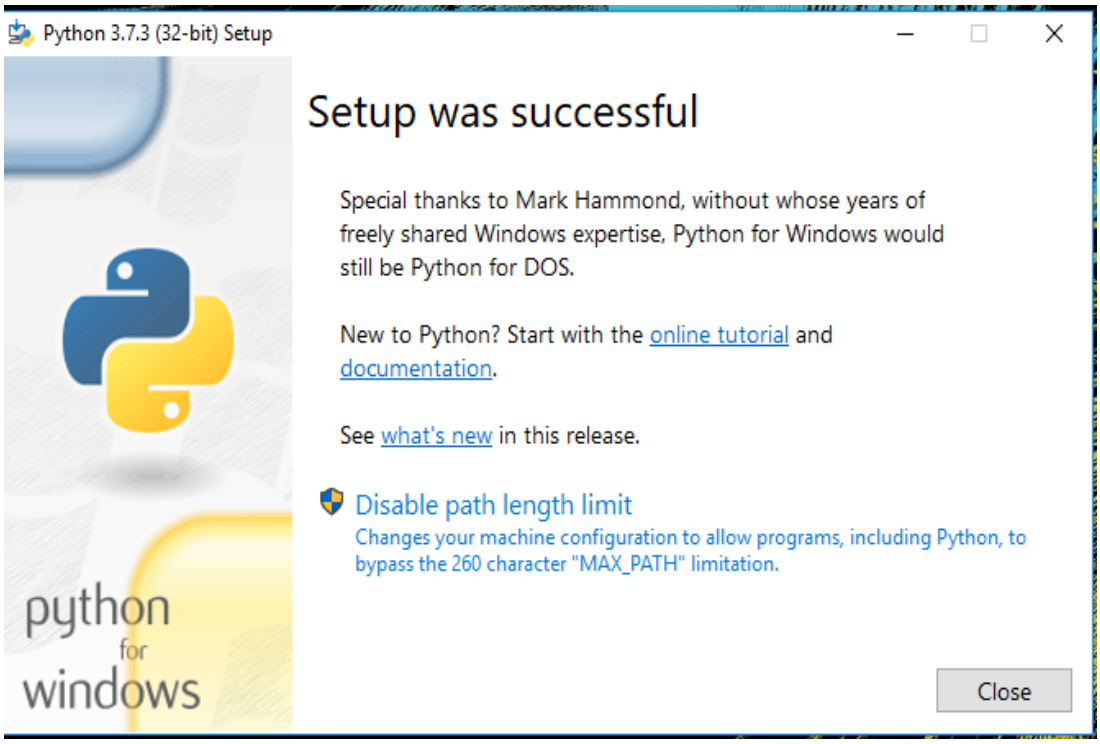
4. This takes you to Advanced Options available while installing Python. Here, select the Install for all users and Add Python to environment variables check boxes.

Optionally, you can select the Associate files with Python, Create shortcuts for installed applications and other advanced options. Make note of the python installation directory displayed in this step. You would need it for the next step.

After selecting the Advanced options, click Install to start installation.



4. Once the installation is over, you will see a Python Setup Successful window.



Step 3: Add Python to environmental variables

The last (optional) step in the installation process is to add Python Path to the System Environment variables. This step is done to access Python through the command line. In case you have added Python to environment variables while setting the Advanced options during the installation procedure, you can avoid this step. Else, this step is done manually as follows.

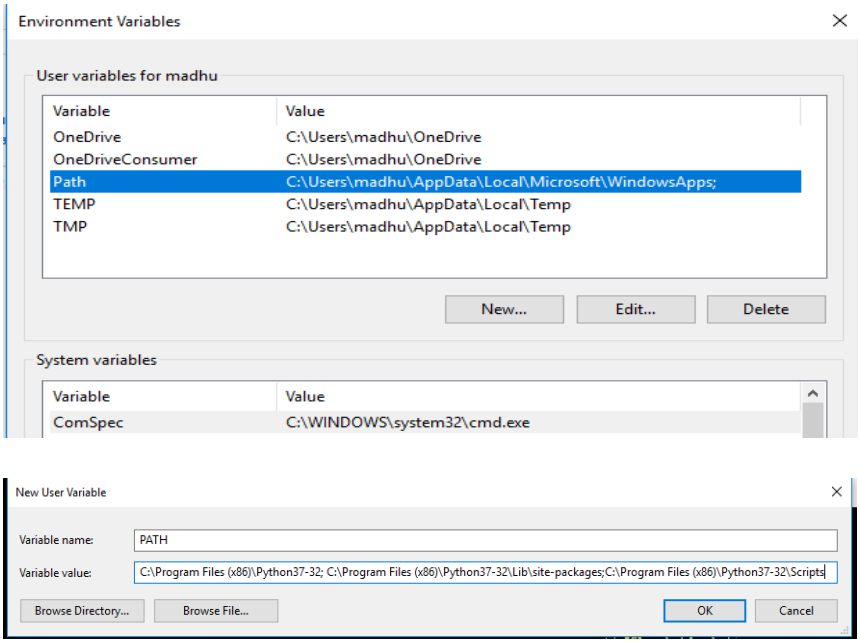
In the Start menu, search for “advanced system settings”. Select “View advanced system settings”. In the “System Properties” window, click on the “Advanced” tab and then click on the “Environment Variables” button.

Locate the Python installation directory on your system. If you followed the steps exactly as above, python will be installed in below locations:

- C:\Program Files (x86)\Python37-32: for 32-bit installation
- C:\Program Files\Python37-32: for 64-bit installation

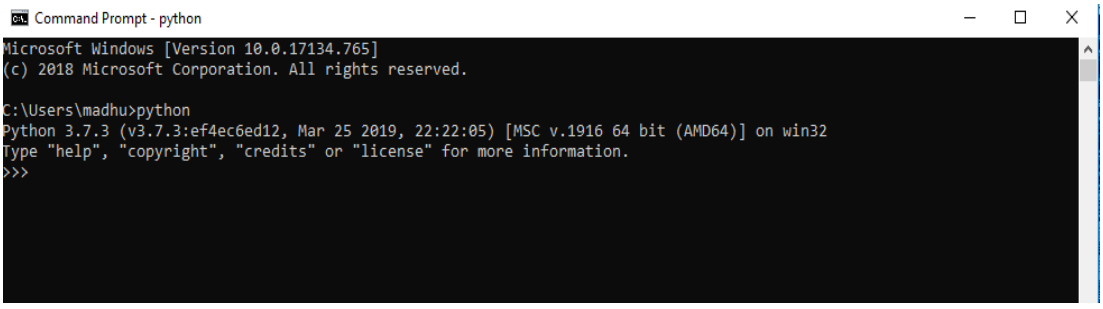
The folder name may be different from “Python37-32” if you installed a different version. Look for a folder whose name starts with Python.

Append the following entries to PATH variable as shown below:

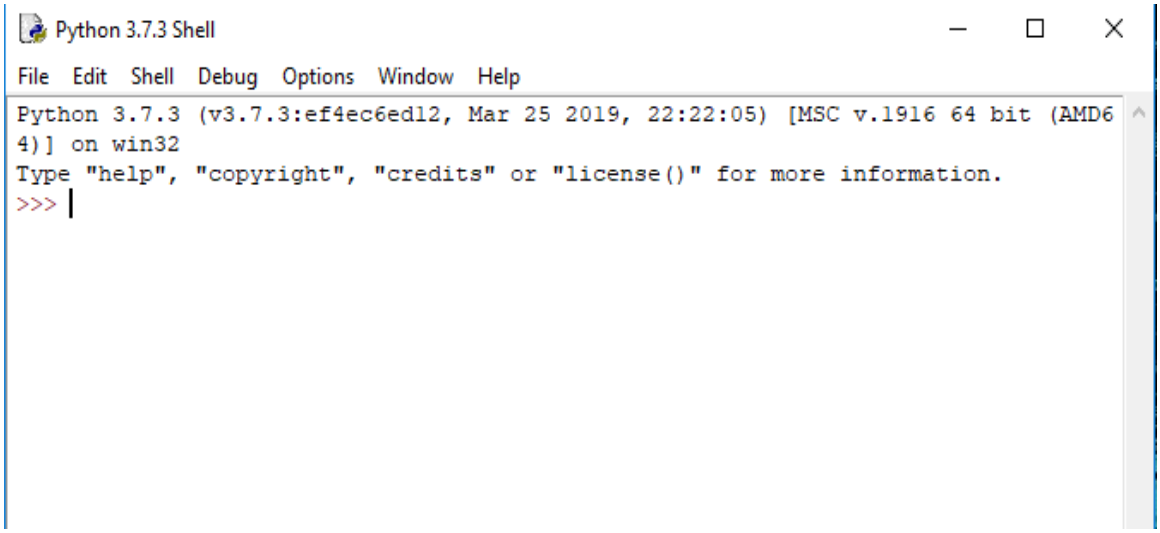


Step 4: Verify the Python Installation

You have now successfully installed Python 3.7.3 on Windows 10. You can verify if the Python installation is successful either through the command line or through the IDLE app that gets installed along with the installation. Search for the command prompt and type “python”. You can see that Python 3.7.3 is successfully installed.



An alternate way to reach python is to search for “Python” in the start menu and clicking on IDLE (Python 3.7 64-bit). You can start coding in Python using the Integrated Development Environment (IDLE).



USES

Since 2003, Python has consistently ranked in the top ten most popular programming languages in the TIOBE Programming Community Index where, as of February 2020, it is the third most popular language (behind Java, and C). It was selected Programming Language of the Year in 2007, 2010, and 2018.

- An empirical study found that scripting languages, such as Python, are more productive than conventional languages, such as C and Java, for programming problems involving string manipulation and search in a dictionary, and determined that memory consumption was often "better than Java and not much worse than C or C++".
- Large organizations that use Python include Wikipedia, Google, Yahoo!, CERN, NASA, Facebook, Amazon, Instagram, Spotify and some smaller entities like ILM and ITA. The social news networking site Reddit is written entirely in Python.
- Python can serve as a scripting language for web applications, e.g., via mod_wsgi for the Apache web server. With Web Server Gateway Interface, a standard API has evolved to facilitate these applications. Web frameworks like Django, Pylons, Pyramid, TurboGears, web2py, Tornado, Flask, Bottle and Zope support developers in the design and maintenance of complex applications. Pyjs and IronPython can be used to develop the client-side of Ajax-based applications.

- SQLAlchemy can be used as data mapper to a relational database. Twisted is a framework to program communications between computers, and is used (for example) by Dropbox.
- Libraries such as NumPy, SciPy and Matplotlib allow the effective use of Python in scientific computing, with specialized libraries such as Bio python and Astropy providing domain-specific functionality.
- SageMath is a mathematical software with a notebook interface programmable in Python: its library covers many aspects of mathematics, including algebra, combinatorics, numerical mathematics, number theory, and calculus.
- Python has been successfully embedded in many software products as a scripting language, including in finite element method software such as Abaqus, 3D parametric modeler like Free CAD, 3D animation packages such as 3ds Max, Blender, Cinema 4D, Lightwave, Houdini, Maya, modo, Motion Builder, Softimage, the visual effects compositor Nuke, 2D imaging programs like GIMP, Inkscape, Scribus and Paint Shop Pro, and musical notation programs like score writer and Capella.
- GNU Debugger uses Python as a pretty printer to show complex structures such as C++ containers. Esri promotes Python as the best choice for writing scripts in ArcGIS.
- It has also been used in several video games, and has been adopted as first of the three available programming languages in Google App Engine, the other two being Java and Go.
- Python is commonly used in artificial intelligence projects with the help of libraries like TensorFlow, Keras, Pytorch and Scikit-learn. As a scripting language with modular architecture, simple syntax and rich text processing tools, Python is often used for natural language processing.
- Many operating systems include Python as a standard component. It ships with most Linux distributions, AmigaOS 4, FreeBSD (as a package), NetBSD, OpenBSD (as a package) and macOS and can be used from the command line (terminal).

- Many Linux distributions use installers written in Python: Ubuntu uses the Ubiquity installer, while Red Hat Linux and Fedora use the Anaconda installer. Gentoo Linux uses Python in its package management system, Portage.
- Python is used extensively in the information security industry, including in exploit development.
- Most of the Sugar software for the One Laptop per Child XO, now developed at Sugar Labs, is written in Python. The Raspberry Pi single-board computer project has adopted Python as its main user-programming language.
- Due to Python's user-friendly conventions and easy-to-understand language, it is commonly used as an intro language into computing sciences with students. This allows students to easily learn computing theories and concepts and then apply them to other programming languages.
- LibreOffice includes Python, and intends to replace Java with Python. Its Python Scripting Provider is a core feature [169] since Version 4.0 from 7 February 2013.

CHAPTER-8
SAMPLE CODE

8.SAMPLE CODE

8.1 IMPORTING FILES

```
from google.colab import files
files.upload()
! pip install  kaggle --upgrade
!mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle
!chmod 600 ~/.kaggle/kaggle.json
! kaggle competitions download -c malware-detection
#! unzip Kaggle-data.csv.zip
! unzip /content/malware-detection.zip
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
```

8.2 CHECKING & CLEANING OF DATA

```
warnings.filterwarnings('ignore')
df=pd.read_csv('Kaggle-data.csv')
df1=df.drop(13) # droping missing value
df1.head()
df1.describe()
df1.info()
df1.isnull().sum()
df1["legitimate"].head()
# df=df.dropna()
y=df1['legitimate']
df1=df1.drop(columns=['Unnamed: 57','legitimate'])
df1.sample(5)
sns.countplot(y)
df1.dtypes
from sklearn.preprocessing import LabelEncoder

encoder=LabelEncoder()
object_cols=df1.select_dtypes(include='object').columns
for i in object_cols:
  df1[i]=encoder.fit_transform(df1[i].astype(str))

df1.info()
df1.sample(5)
```

8.3 CORRELATION BETWEEN VARIABLES

```
from sklearn.preprocessing import MinMaxScaler

scaler=MinMaxScaler()
cols=df1.columns
data_scaled=scaler.fit_transform(df1.values)

df_scaled=pd.DataFrame(data=data_scaled,columns=cols)

df_scaled.sample(5)
fig=plt.figure(figsize=(16,12))
sns.heatmap(df1.corr())
plt.title('Correlation between variables')
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test=train_test_split(df_scaled,y,
test_size=0.25,random_state=42,stratify=y)
print(f'X_trains shape is {X_train.shape}, y_train shape is
 {y_train.shape}')
```

8.4 LEGITIMATE VALUES

```
import time
begin = time.time()
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
rf=RandomForestClassifier()
rf.fit(X_train,y_train)
rf_acc=rf.score(X_test,y_test)

y_rf_pred=rf.predict(X_test)
time.sleep(1)
end = time.time()
time_rf=end - begin
print('Confusion Matrix ')
arr=(confusion_matrix(y_test,y_rf_pred))
# print(arr)
df_rf = pd.DataFrame(arr, ['legitimate','illegitimate'], ['
legitimate','illegitimate'])
# plt.figure(figsize=(10,7))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_rf, annot=True, annot_kws={"size": 16},fmt='
g') # font size
```

8.5 RANDOM FOREST CLASSIFIER

```
plt.show()

print('Random Forest classifier accuracy is :',rf_acc)
print(f"Total runtime of the program is {time_rf}")
print(classification_report(y_test,y_rf_pred))
import time
begin = time.time()
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report
gb=GradientBoostingClassifier()

gb.fit(X_train,y_train)

gb_acc=gb.score(X_test,y_test)

y_gb_pred=gb.predict(X_test)
time.sleep(1)
end = time.time()
time_gbc=end - begin
print('Confusion Matrix ')
arr1=(confusion_matrix(y_test,y_gb_pred))
# print(arr)
df_gb = pd.DataFrame(arr1, ['legitimate','illegitimate'], [
'legitimate','illegitimate'])
# plt.figure(figsize=(10,7))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_gb, annot=True, annot_kws={"size": 16},fmt='
g') # font size
```

8.6 GRADIENT BOOSTING CLASSIFIER

```
plt.show()

print('Gradient boosting classifier accuracy is :',gb_acc)
print(f"Total runtime of the program is {time_gbc}")
print(classification_report(y_test,y_gb_pred))
import time
begin = time.time()
import xgboost as xgb
from sklearn.metrics import classification_report
xgb_clf=xgb.XGBClassifier()
xgb_clf.fit(X_train,y_train)
xgb_acc=xgb_clf.score(X_test,y_test)

y_xgb_pred=xgb_clf.predict(X_test)
```

```
time.sleep(1)
end = time.time()
time_xgb=end - begin
print('Confusion Matrix ')
arr2=(confusion_matrix(y_test,y_xgb_pred))
# print(arr)
df_xgb = pd.DataFrame(arr2, ['legitimate','illegitimate'],
['legitimate','illegitimate'])
# plt.figure(figsize=(10,7))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_xgb, annot=True, annot_kws={"size": 16},fmt=
'g') # font size
```

8.7 XGBOOST CLASSIFIER

```
plt.show()

print('XGBoost classifier accuracy is :',xgb_acc)
print(f"Total runtime of the program is {time_xgb}")

print(classification_report(y_test,y_xgb_pred))
import time
begin = time.time()
from sklearn.svm import SVC
from sklearn.metrics import classification_report
svc=SVC()
svc.fit(X_train,y_train)
svm_acc=svc.score(X_test,y_test)
y_svm_pred=svc.predict(X_test)
time.sleep(1)
end_svm = time.time()
time_svm=end_svm - begin
print('Confusion Matrix ')

arr4=(confusion_matrix(y_test,y_svm_pred))
# print(arr)
df_svm = pd.DataFrame(arr4, ['legitimate','illegitimate'],
['legitimate','illegitimate'])
# plt.figure(figsize=(10,7))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_svm, annot=True, annot_kws={"size": 16},fmt=
'g') # font size
```

8.8 SVM

```
plt.show()

print('SVM accuracy is :',svm_acc)
print(f"Total runtime of the program is {time_svm}")
print(classification_report(y_test,y_svm_pred))
import time
begin = time.time()
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
NB=GaussianNB()
NB.fit(X_train,y_train)
nb_acc=NB.score(X_test,y_test)
y_nb_pred=NB.predict(X_test)
time.sleep(1)
end_nb = time.time()
time_nb=end_nb - begin
print('Confusion Matrix ')
arr5=(confusion_matrix(y_test,y_nb_pred))
# print(arr)
# print(type(arr))
# arr=arr.astype('float64')
df_nb = pd.DataFrame(arr5, ['yes','No'], ['yes','No'])
# plt.figure(figsize=(10,7))
sns.set(font_scale=1.4) # for label size

sns.heatmap(df_nb, annot=True, annot_kws={"size": 16},fmt='
g') # font size
```

8.9 NAÏVE BAYES

```
plt.show()

print('Naive Bayes Classifier Accuracy :',nb_acc)
print(f"Total runtime of the program is {time_nb}")
print(classification_report(y_test,y_nb_pred))
begin = time.time()
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
lr=LogisticRegression()
lr.fit(X_train,y_train)
lr_acc=lr.score(X_test,y_test)

y_lr_pred=lr.predict(X_test)
```

```
time.sleep(1)
end = time.time()
time_lr=end - begin
print('Confusion Matrix ')
arr=(confusion_matrix(y_test,y_lr_pred))
# print(arr)
df_lr = pd.DataFrame(arr, ['legitimate','illegitimate'], ['
legitimate','illegitimate'])
# plt.figure(figsize=(10,7))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_lr, annot=True, annot_kws={"size": 16},fmt='
g') # font size
```

8.10 LOGISTIC REGRESSION

```
plt.show()

print('Logistic Regression accuracy is :',lr_acc)
print(f"Total runtime of the program is {time_lr}")
print(classification_report(y_test,y_lr_pred))
import time
begin = time.time()
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
dct=DecisionTreeClassifier()
dct.fit(X_train,y_train)
dct_acc=dct.score(X_test,y_test)

y_dct_pred=dct.predict(X_test)
time.sleep(1)
end = time.time()

time_dct=end - begin
print('Confusion Matrix ')
arr=(confusion_matrix(y_test,y_dct_pred))
# print(arr)
df_dct = pd.DataFrame(arr, ['legitimate','illegitimate'], [
'legitimate','illegitimate'])
# plt.figure(figsize=(10,7))
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_dct, annot=True, annot_kws={"size": 16},fmt=
'g') # font size
```

8.11 DECISION TREE

```
plt.show()

print('Decision Tree classifier accuracy is :',dct_acc)
print(f"Total runtime of the program is {time_dct}")
print(classification_report(y_test,y_dct_pred))
res=pd.DataFrame(data=[[rf_acc,svm_acc,gb_acc,xgb_acc,nb_ac
c,lr_acc,dct_acc]],columns=['Random Forest','SVM','Gradient
 boost','XGbosst','Naive-
Bayes','Logistic Regression','Decision Tree'])
fig=plt.figure(figsize=(16,6))
sns.barplot(data=res)
```

8.12 COMPARISION OF MODELS

```
plt.title('Comparision of model Accuracies')
plt.xlabel('Models')
plt.ylabel('Accuracy')
res=pd.DataFrame(data=[[time_rf,time_svm,time_gbc,time_xgb,
time_nb,time_lr,time_dct]],columns=['Random Forest','SVM','
Gradient boost','XGbosst','Naive-
Bayes','Logistic Regression','Decision Tree' ])
fig=plt.figure(figsize=(16,6))
sns.barplot(data=res)
plt.title('Comparision of model Time of Execution')
plt.xlabel('Models')
plt.ylabel('Time Taken(in sec)')
```

CHAPTER-9
SYSTEM TESTING

9.SYSTEM TESTING

9.1 INTRODUCTION

The motivation behind testing is to find lapses. Testing is the procedure of attempting to find each possible blame or shortcoming in a work item. It gives an approach to check the usefulness of parts, sub get-togethers, gatherings and/or a completed item It is the methodology of practicing programming with the aim of guaranteeing that the Programming framework lives up to its prerequisites and client desires and does not fizzle in an unsuitable way. There are different sorts of test. Every test sort addresses a particular testing necessity.

DESIGN OF TEST CASES & SCENARIOS

The main objective of testing is to uncover a host of errors, systematically and with minimum effort and time. Stating formally, we can say, testing is a process of executing a program with intent of finding an error.

CODE TESTING

This examines the logic of the program. For example, the logic for updating various sample data and with the sample files and directories were tested and verified.

SPECIFICATION TESTING

Executing this specification starting what the program should do and how it should performed under various conditions. Test cases for various situation and combination of conditions in all the modules are tested.

UNIT TESTING

In the unit testing we test each module individually and integrate with the overall system. Unit testing focuses verification efforts on the smallest unit of software design in the module. This is also known as module testing. The module of the system is tested separately. This testing is carried out during programming stage itself. In the testing step each module is found to work satisfactorily as regard to expected output from the module. There are some validation checks for fields also. For example the validation check is done for varying the user input given by the user which validity of the data entered. It is very easy to find error debut the system.

Each Module can be tested using the following two Strategies:

1. Black Box Testing

2. White Box Testing

BLACK BOX TESTING

Black box testing is a software testing techniques in which **functionality of the software under test (SUT) is tested without looking at the internal code structure**, implementation details and knowledge of internal paths of the software. This type of testing is based entirely on the software requirements and specifications.

BLACK BOX TESTING – STEPS

Here are the generic steps followed to carry out any type of Black Box Testing.

- Initially prerequisites and details of the framework are inspected.
- Tester picks substantial inputs (positive test situation) to check whether SUT forms them accurately. Additionally, some invalid inputs (negative test situation) are decided to confirm that the SUT has the capacity identify them.
- Tester decides expected yields for each one of those inputs.
- Software analyser develops experiments with the chose inputs.
- The experiments are executed.
- Software analyser contrasts the real yields and the normal yields.
- Defects if any are altered and re-tried.

There are numerous sorts of Black Box Testing yet taking after are the conspicuous ones –

- **Functional testing** – This discovery testing sort is identified with practical prerequisites of a framework; it is finished by programming analyzers.
- **Non-practical testing** – This sort of discovery testing is not identified with testing of a particular usefulness, but rather non-useful prerequisites, for example, execution, adaptability, ease of use.

- **Regression testing** – Regression testing is done after code fixes; redesigns or some other framework upkeep to check the new code has not influenced the current code.

WHITE BOX TESTING

White Box Testing is the trying of a product arrangement's interior coding and base. It concentrates basically on reinforcing security, the stream of inputs and yields through the application, and enhancing configuration and ease of use.

What do we verify in White Box Testing?

White box testing involves the testing of the software code for the following:

- Internal security holes
- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object and function on an individual basis

The testing can be done at system, integration and unit levels of software development. One of the basic goals of white box testing is to verify a working flow for an application. It involves testing a series of predefined inputs against expected or desired outputs so that

When a specific input does not result in the expected output, you have encountered a bug.

How do we perform White Box Testing?

To give you a simplified explanation of white box testing, we have divided it into two basic steps. This is what testers do when testing an application using the white box Testing technique.

Step 1) Understand The Source Code

The first thing a tester will usually do is learn and understand the source code of the application. Since white box testing involves the testing of the inner workings of an application, the tester must be very knowledgeable in the programming languages used

in the applications they are testing. Also, the testing person must be highly aware of secure coding practices. Security is often one of the primary objectives of

testing software. The tester should be able to find security issues and prevent attacks from hackers and naïve users WHO might inject malicious code into the application either knowingly or unknowingly.

Step 2) Create Test Cases and Execute

The second basic step to white box testing involves testing the application's source code for proper flow and structure. One way is by writing more code to test the application's source code. The tester will develop little tests for each process or series of processes in the application. This method requires that the tester must have intimate knowledge of the code and is often done by the developer. Other methods include manual testing, trial and error testing and the use of testing tools as we will explain further on in this article.

9.2 SYSTEM TESTING

Once the individual module testing is completed, modules are assembled and integrated to perform as a system. The top down testing, which began from upper level to lower level module, was carried out to check whether the entire system is performing satisfactorily. There are three main kinds of System testing: Alpha Testing, Beta Testing and Acceptance Testing.

ALPHA TESTING

This refers to the system testing that is carried out by the test team within the Organization.

BETA TESTING

This refers to the system testing that is performed by a selected group of friendly customers.

ACCEPTANCE TESTING

This refers to the system testing that is performed by the customer to determine whether or not to accept the delivery of the system.

INTEGRATION TESTING

Data can be lost across an interface, one module can have an adverse effort on the other sub functions, when combined, may not produce the desired major functions. Integrated testing is the systematic testing for constructing the uncover errors within the interface.

The testing was done with sample data. The developed system has run successfully for this sample data. The need for integrated test is to find the overall system performance.

OUTPUT TESTING

After performance of the validation testing, the next step is output testing. The output displayed or generated by the system under consideration is tested by asking the user about the format required by system. The output format on the screen is found to be correct as format was designed in the system phase according to the user needs. Hence the output testing does not result in any correction in the system.

TEST PLAN

The test-plan is basically a list of test cases that need to be run on the system. Some of the test cases can be run independently for some components (report generation from the database, for example, can be tested independently) and some of the test cases require the whole system to be ready for their execution. it is better to test each component as and when it's ready before integrating the components. it's important to note that the test cases cover all the aspects of the system (ie, all the requirements stated in the RS document).

VERIFICATION AND VALIDATION

We have mentioned all the test cases applicable for the application we are developing. They must go through the test process and make sure they all pass. All these results will be recorded for the metrics purposes. Validation checks that the product design satisfies or fits the intended use (high-level checking), i.e., the software meets the user requirements. This is done through dynamic testing and other forms of review. Verification and validation are not the same thing, although they are often confused. Succinctly expressed the difference between them:

Verification: The procedure of assessing programming to figure out if the results of a given improvement stage fulfil the conditions forced toward the begin of that stage.

Acceptance: The procedure of assessing programming amid or toward the end of the improvement procedure to figure out if it fulfils indicated necessities. In other words, validation guarantees that the item really addresses the client's issues, and that the determinations were right in any case, while confirmation is guaranteeing that the item has been assembled by necessities and configuration details. Acceptance guarantees that "you fabricated the correct thing". Confirmation guarantees that "you constructed it right". Acceptance affirms that the item, as gave, will satisfy its proposed utilization. From testing viewpoint:

Deficiency – wrong or missing capacity in the code.

Disappointment – the sign of a deficiency amid execution.

Breakdown – as per its particular the framework does not meet its predefined usefulness. Inside of the demonstrating and reproduction group, the meanings of acceptance,

CONFIRMATION AND ACCREDITATION ARE COMPARATIVE:

Acceptance is the procedure of deciding the extent to which a model, reproduction, or organization of models and recreations, and their related information are exact representations of this present reality from the viewpoint of the expected use(s).

Accreditation is the formal affirmation that a model or reproduction is adequate to be utilized for a particular reason.

Confirmation is the procedure of verifying that a PC model, reproduction, or alliance of models and re-enactments executions and their related information precisely speak to the engineer's reasonable depiction and determination**.**

9.3 TESTING METHODOLOGIES

The following are the Testing Methodologies:

- **Unit Testing.**
- **Integration.**
- **User Acceptance.**
- **Output Testing.**
- **Validation.**

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.

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 a 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.

9.4 TESTING STRATAGY

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-10

SCREEN SHOTS

10.SCREEN SHOTS

10.1 LEGITIMATE COUNT PLOTS

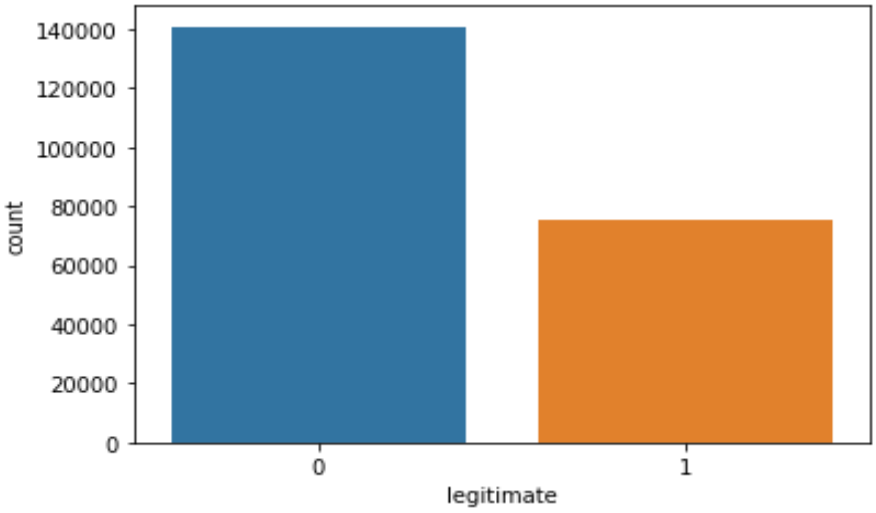


FIG 10.1 LEGITIMATE COUNT PLOTS

10.2 CORRELATION BETWEEN VARIABLES

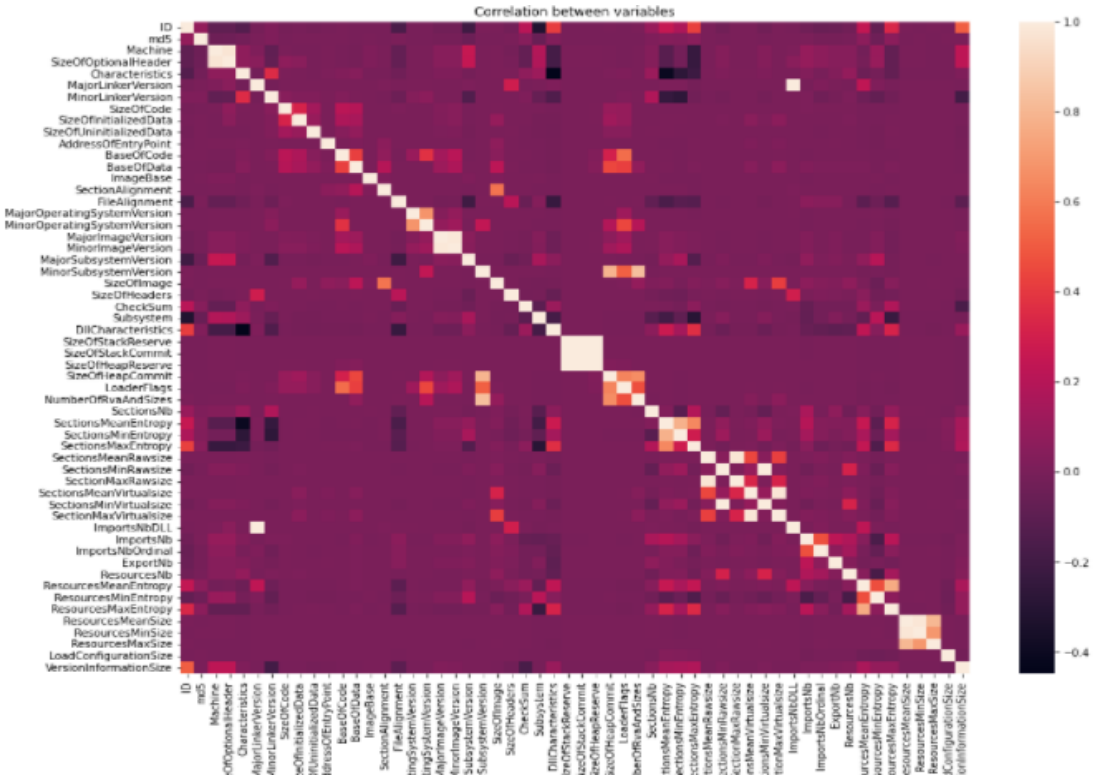
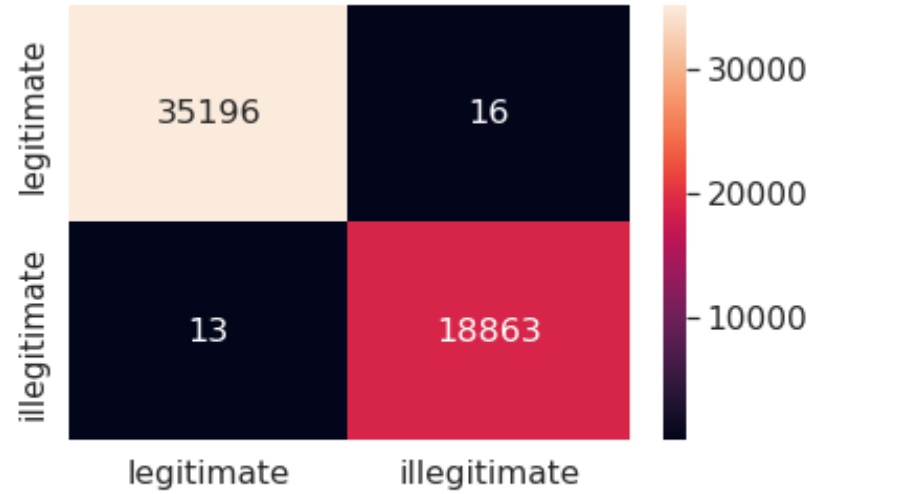


FIG 10.2 CORRELATION BETWEEN VARIABLES

10.3 CONFUSION MATRIX FOR RANDOM FOREST




```
Random Forest classifier accuracy is : 0.9994638367105457
Total runtime of the program is 50.65211844444275
```

FIG10.3 CONFUSION MATRIX FOR RANDOM FOREST

10.4 CONFUSION MATRIX FOR SVM

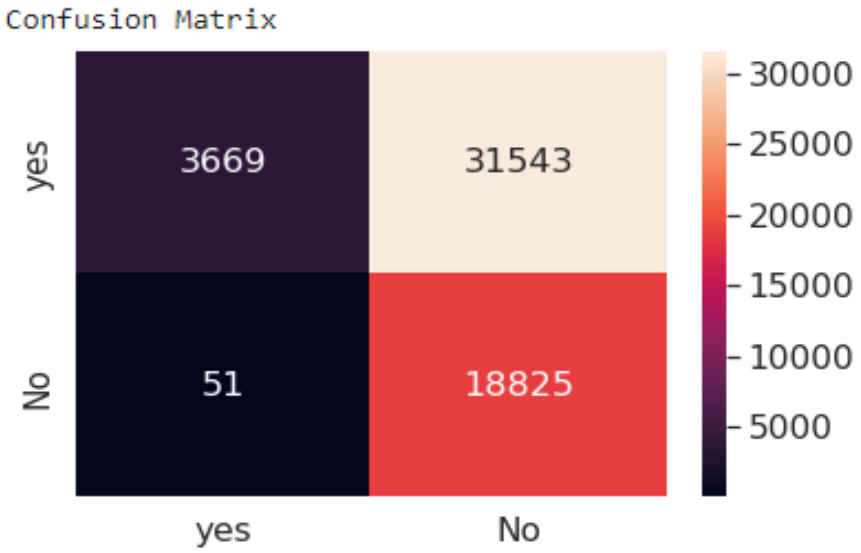

```
SVM accuracy is : 0.9926970862298476
Total runtime of the program is 316.86604833602905
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     35212
           1       0.99      0.99      0.99     18876

    accuracy                           0.99     54088
   macro avg       0.99      0.99      0.99     54088
weighted avg       0.99      0.99      0.99     54088
```

FIG 10.4 CONFUSION MATRIX FOR SVM

10.5 CONFUSION MATRIX FOR NAÏVE BAYES CLASSIFIER



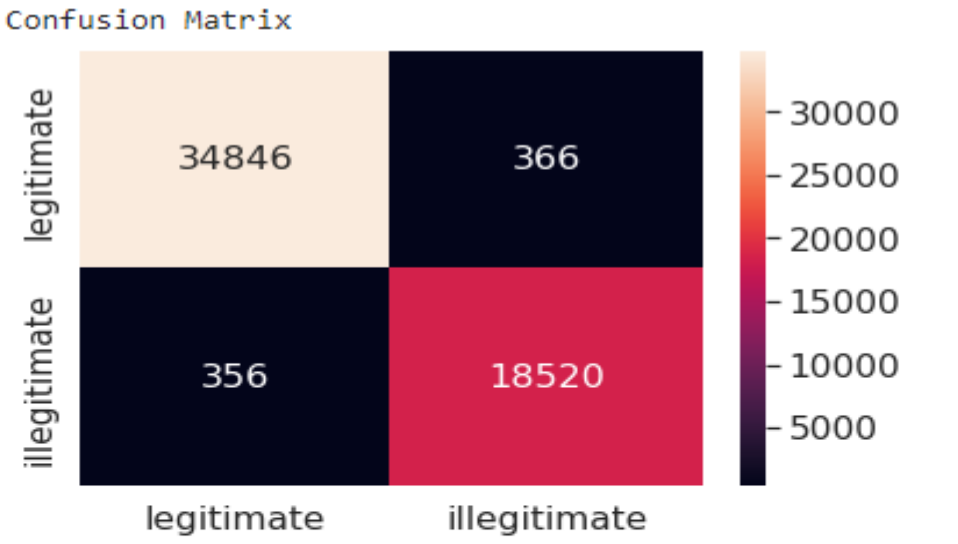

```
Naive Bayes Classifier Accuracy : 0.4158778287235616
Total runtime of the program is 1.2696080207824707
              precision    recall  f1-score   support

           0       0.99      0.10      0.19     35212
           1       0.37      1.00      0.54     18876

    accuracy                           0.42     54088
   macro avg       0.68      0.55      0.37     54088
weighted avg       0.77      0.42      0.31     54088
```

FIG 10.5 CONFUSION MATRIX FOR NAÏVE BAYES CLASSIFIER

10.6 CONFUSION MATRIX FOR LOGISTIC REGRESSION

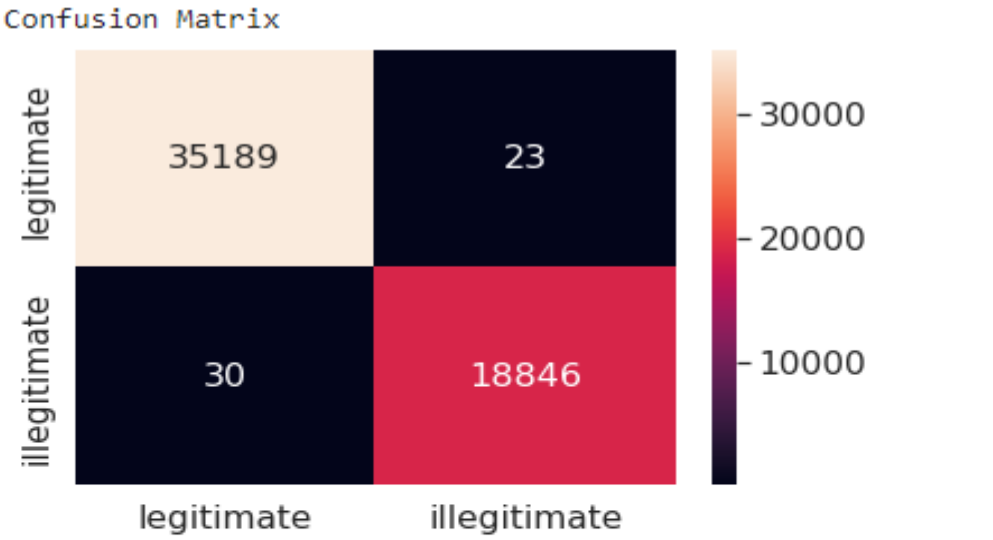


Logistic Regression accuracy is : 0.986651382931519
Total runtime of the program is 4.972764730453491

	precision	recall	f1-score	support
0	0.99	0.99	0.99	35212
1	0.98	0.98	0.98	18876
accuracy			0.99	54088
macro avg	0.99	0.99	0.99	54088
weighted avg	0.99	0.99	0.99	54088

FIG 10.6 CONFUSION MATRIX FOR LOGISTIC REGRESSION

10.7 CONFUSION MATRIX FOR DECISION TREE



Decision Tree classifier accuracy is : 0.9990201153675492
Total runtime of the program is 7.969463586807251

	precision	recall	f1-score	support
0	1.00	1.00	1.00	35212
1	1.00	1.00	1.00	18876
accuracy			1.00	54088
macro avg	1.00	1.00	1.00	54088
weighted avg	1.00	1.00	1.00	54088

FIG 10.7 CONFUSION MATRIX FOR DECISION TREE

10.8 COMPARISION OF MODEL ACCURACIES

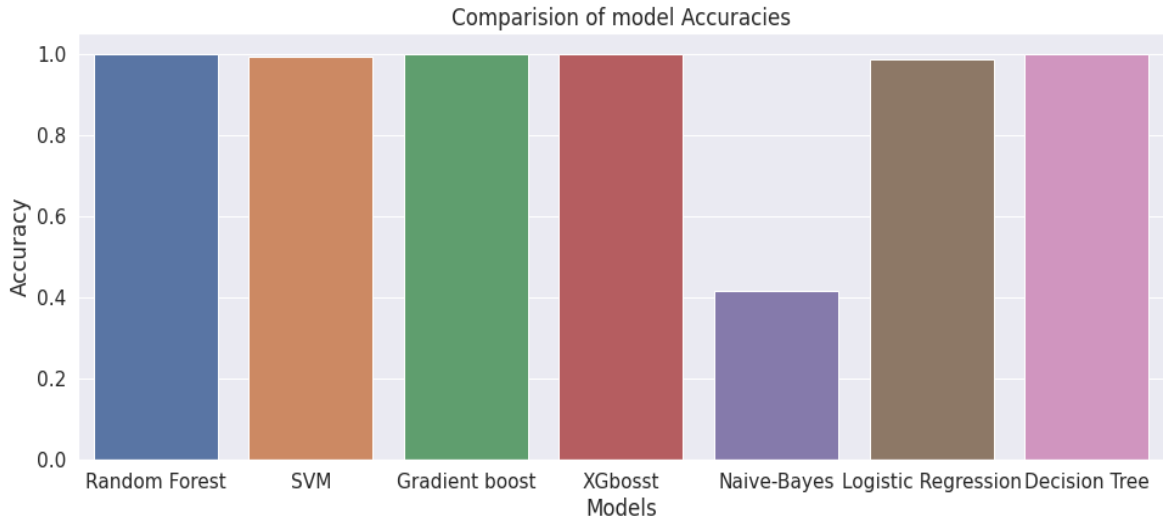


FIG 10.8 COMPARISION OF MODEL ACCURACIES

10.9 COMPARISION OF MODEL TIME OF EXECUTION

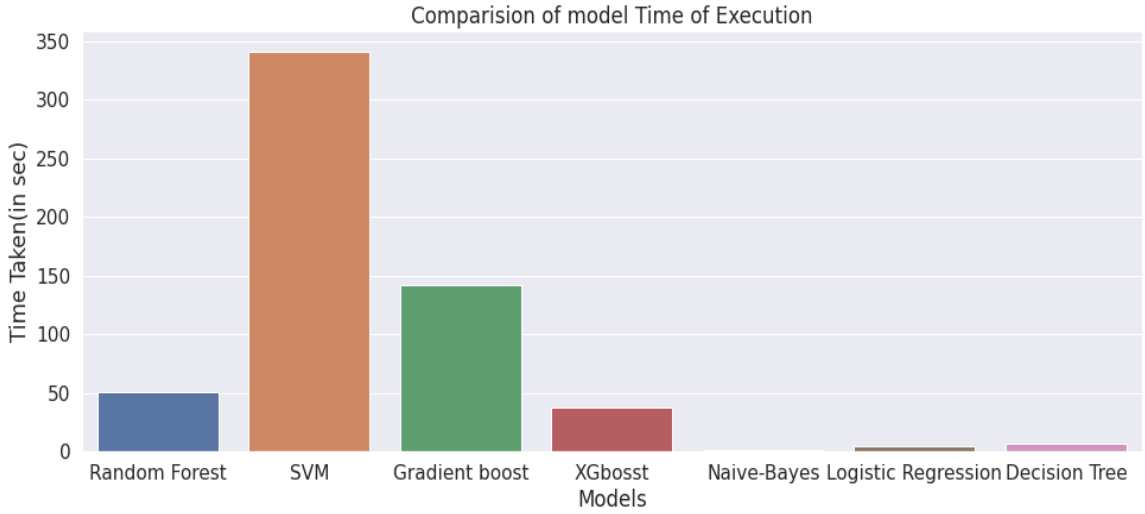


FIG 10.8 COMPARISION OF MODEL TIME OF EXECUTION

CHAPTER-11
CONCLUSION

11.CONCLUSION

The integration of complex cyber physical infrastructures and applications in a CSC environment have brought economic, business, and societal impact for both national and global context in the areas of Transport, Energy, Healthcare, Manufacturing, and Communication. However, CPS security remains a challenge as vulnerability from any part of the system can pose risk within the overall supply chain context. This paper aims to improve CSC security by integrating CTI and ML for the threat analysis and predication. We considered the necessary concepts from CSC and CTI and a systematic process to analyse and predicate the threat. The experimental results showed that accuracies of the LG, DT, SVM, and RF algorithms in Majority Voting and identified a list of predicated threats. We also observed that CTI is effective to extract threat information, which can integrate into the ML classifiers for the threat predication. This allows CSC organization to analyse the existing controls and determine additional controls for the improvement of overall cyber security. It is necessary to consider the full automation of the process and industrial case study to generalize our findings. Furthermore, we are also planning to consider evaluating the existing controls and the necessary of future controls based on our prediction results.

CHAPTER-11
REFERENCES

12.REFERENCES

[1] National Cyber Security Centre. (2018). Example of Supply Chain Attacks. [Online] Available: https://www.ncsc.gov.uk/collection/supplychain-security/supply-chain-attack-examples

[2] A. Yeboah-Ofori and S. Islam, ‘‘Cyber security threat modelling for supply chain organizational environments,’’ MDPI. Future Internet, vol. 11, no. 3, p. 63, Mar. 2019. [Online]. Available: https://www.mdpi.com/1999- 5903/11/3/63

[3] B. Woods and A. Bochman, ‘‘Supply chain in the software era,’’ in Scowcroft Center for Strategic and Security. Washington, DC, USA: Atlantic Council, May 2018.

[4] Exploring the Opportunities and Limitations of Current Threat Intelligence Platforms, Version 1, ENISA, Dec. 2017. [Online]. Available: https://www.enisa.europa.eu/publications/exploring-the-opportunitiesand-limitations-of-current- threat-intelligence-platforms

[5] C. Doerr, TU Delft CTI Labs. (2018). Cyber Threat Intelligences Standards—A High Level Overview. [Online]. Available: https://www. enisa.europa.eu/events/2018-cti-eu-event/cti-eu-2018-presentations/ cyber-threat-intelligence-standardization.pdf

[6] Research Prediction. (2019). Microsoft Malware Prediction. [Online]. Available: https://www.kaggle.com/c/microsoft-malware-prediction/data

[7] A. Yeboah-Ofori and F. Katsriku, ‘‘Cybercrime and risks for cyber physical systems,’’ Int. J. Cyber-Secur. Digit. Forensics, vol. 8, no. 1, pp. 43–57, 2019.

[8] CAPEC-437, Supply Chain. (Oct. 2018). Common Attack Pattern Enumeration and Classification: Domain of Attack. [Online]. Available: https://capec.mitre.org/data/definitions/437.html

[9] Open Web Application Security Project (OWASP). (2017). The Ten Most Critical Application Security Risks, Creative Commons Attribution-Share Alike 4.0 International License. [Online] Available: https://owasp.org/ www-pdf-archive/OWASP_Top_10-2017_%28en%29.pdf.pdf

[10] US-Cert. (2020). Building Security in Software & Supply Chain Assurance. [Online]. Available: https://www.us-cert.gov/bsi/articles/ knowledge/attack-patterns

[11] R. D. Labati, A. Genovese, V. Piuri, and F. Scotti, ‘‘Towards the prediction of renewable energy unbalance in smart grids,’’ in Proc. IEEE 4th Int. Forum Res. Technol. Soc. Ind. (RTSI), Palermo, Italy, Sep. 2018, pp. 1–5, doi: 10.1109/RTSI.2018.8548432.

[12] J. Boyens, C. Paulsen, R. Moorthy, and N. Bartol, ‘‘Supply chain risk management practices for federal information systems and organizations,’’ NIST Comput. Sec., vol. 800, no. 161, p. 32, 2015, doi: 10.6028/NIST.SP.800-161.

[13] Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1, NIST, Gaithersburg, MD, USA, 2018, doi: 10.6028/NIST.CSWP.04162018.

[14] J. F. Miller, ‘‘Supply chain attack framework and attack pattern,’’ MITRE, Tech. Rep. MTR140021, 2013. [Online]. Available: https://www. mitre.org/sites/default/files/publications/supply-chain-attack-framework14-0228.pdf

[15] C. Ahlberg and C. Pace. The Threat Intelligence Handbook. [Online]. Available: https://paper.bobylive.com/Security/threat-intelligencehandbook-second-edition.pdf

[16] J. Freidman and M. Bouchard, ‘‘Definition guide to cyber threat intelligence. Using knowledge about adversary to win the war against targeted attacks,’’ iSightPartners, CyberEdge Group LLC, Annapolis, MD, USA, Tech. Rep., 2018. [Online]. Available: https://cryptome.org/2015/09/ctiguide.pdf

[17] EY. (2016). Cyber Threat I1ntelligence: Designing, Building and Operating an Effective Program. [Online]. Available: https://relayto.com/eyfrance/cyber-threat-intelligence-report-js5wmwy7/pdf

[18] A. Yeboah-Ofori and C. Boachie, ‘‘Malware attack predictive analytics in a cyber supply chain context using machine learning,’’ in Proc. ICSIoT, 2019, pp. 66–73, doi: 10.1109/ICSIoT47925.2019.00019.

[19] B. Gallagher and T. Eliassi-Rad, ‘‘Classification of HTTP attacks: A study on the ECML/PKDD 2007 discovery challenge,’’ Lawrence Liverpool Nat. Lab., Livermore, CA, USA, Tech. Rep., 2009, doi: 10.2172/1113394.

[20] D. Bhamare, T. Salman, M. Samaka, A. Erbad, and R. Jain, ‘‘Feasibility of supervised machine learning for cloud security,’’ in Proc. Int. Conf. Inf. Sci. Secur. (ICISS), Dec. 2016, pp. 1–5, doi: 10.1109/ICISSEC.2016.7885853.

[21] A. L. Buczak and E. Guven, ‘‘A survey of data mining and machine learning methods for cyber security intrusion detection,’’ IEEE Commun. Surveys Tuts., vol. 18, no. 2, pp. 1153–1176, 2nd Quart., 2016, doi: 10.1109/COMST.2015.2494502.

[22] O. Yavanoglu and M. Aydos, ‘‘A review on cyber security datasets for machine learning algorithms,’’ in Proc. IEEE Int. Conf. Big Data (Big Data), Dec. 2017, pp. 2186–2193, doi: 10.1109/BigData.2017.8258167.

[23] E. G. V. Villano, ‘‘Classification of logs using machine learning,’’ M.S. thesis, Dept. Inf. Secur. Commun. Technol., Norwegian Univ. Sci. Technol., Trondheim, Norway, 2018.

[24] R. C. B. Hink, J. M. Beaver, M. A. Buckner, T. Morris, U. Adhikari, and S. Pan, ‘‘Machine learning for power system disturbance and cyber-attack discrimination,’’ in Proc. 7th Int. Symp. Resilient Control Syst. (ISRCS), Denver, CO, USA, Aug. 2014, pp. 1–8, doi: 10.1109/ISRCS.2014.6900095.

[25] A. Gumaei, M. M. Hassan, S. Huda, M. R. Hassan, D. Camacho, J. D. Ser, and G. Fortino, ‘‘A robust cyberattack detection approach using optimal features of SCADA power systems in smart grids,’’ Appl. Soft Comput., vol. 96, Nov. 2020, Art. no. 106658, doi: 10.1016/j.asoc. 2020.106658.

[26] M. M. Hassan, A. Gumaei, S. Huda, and A. Almogren, ‘‘Increasing the trustworthiness in the industrial IoT networks through a reliable cyberattack detection model,’’ IEEE Trans. Ind. Informat., vol. 16, no. 9, pp. 6154–6162, Sep. 2020, doi: 10.1109/TII.2020.2970074.

[27] J. Abawajy, S. Huda, S. Sharmeen, M. M. Hassan, and A. Almogren, ‘‘Identifying cyber threats to mobile-IoT applications in edge computing paradigm,’’ Elsevier Sci.

Direct Future Gener. Comput. Syst., vol. 89, pp. 525–538, Dec. 2018, doi: 10.1016/j.future.2018.06.053.

[28] M. M. Rashid, J. Kamruzzaman, M. M. Hassan, T. Imam, and S. Gordon, ‘‘Cyberattacks detection in IoT-based smart city applications using machine learning techniques,’’ Int. J. Environ. Res. Public Health, vol. 17, no. 24, p. 9347, Dec. 2020, doi: 10.3390/ijerph17249347.

[29] M. M. Hassan, S. Huda, S. Sharmeen, J. Abawajy, and G. Fortino, ‘‘An adaptive trust boundary protection for IIoT networks using deep-learning feature-extraction-based semisupervised model,’’ IEEE Trans. Ind. Informat., vol. 17, no. 4, pp. 2860–2870, Apr. 2021, doi: 10.1109/TII.2020.3015026.

[30] M. M. Hassan, M. R. Hassan, S. Huda, and V. H. C. de Albuquerque, ‘‘A robust deep-learning-enabled trust-boundary protection for adversarial industrial IoT environment,’’ IEEE Internet Things J., vol. 8, no. 12, pp. 9611–9621, Jun. 2021, doi: 10.1109/JIOT.2020.3019225.

[31] A. Mohasseb, B. Aziz, J. Jung, and J. Lee, ‘‘Predicting cybersecurity incidents using machine learning algorithms: A case study of Korean SMEs,’’ in Proc. INSTICC, 2019, pp. 230–237, doi: 10.5220/ 0007309302300237.

[32] L. Bilge, Y. Han, and M. D. Amoco, ‘‘Risk teller: Predicting the risk of cyber incidents,’’ in Proc. CCS, 2017, pp. 1299–1311, doi: 10.1145/3133956.3134022.

[33] Y. Liu, A. Sarabi, J. Zhang, P. Naghizadeh, M. Karir, and M. Liu, ‘‘Cloud with a chance of breach: Forecasting cyber security incidents,’’ in Proc. 24th USENIX Secur. Symp., Washington, DC, USA, 2015, pp. 1009–1024.

[34] Guide to Cyber Threat Information Sharing, document NIST 800-150, 2018, doi: 10.6028/NIST.SP.800-150.

[35] S. Barnum, ‘‘Standardizing cyber threat intelligence information with the structured threat information expression,’’ V1.1. Revision, STIX, USA, Tech. Rep., 2014, vol. 1. [Online]. Available: https://www.mitre.org/ publications/technical-papers/standardizing-cyber-threat-intelligenceinformation-with-the

[36] A. Yeboah-Ofori, S. Islam, and E. Yeboah-Boateng, ‘‘Cyber threat intelligence for improving cyber supply chain security,’’ in Proc. Int. Conf. Cyber Secur. Internet Things (ICSIoT), May 2019, pp. 28–33, doi: 10.1109/ICSIoT47925.2019.00012.

[37] A. Boschetti and L. Massaron, Python Data Science Essentials, 2nd ed. Dordrecht, The Netherlands: Springer, 2016. [Online]. Available: https://link.springer.com/content/pdf/10.1007/BF00994018.

A. Yeboah-Ofori, ‘‘Classification of malware attacks using machine learning in decision tree,’’ IJS, vol. 11, no. 2, pp. 10–25, 2020. [Online]. Available: https://www.cscjournals.org/manuscript/Journals/ IJS/Volume11/Issue2/IJS-155.pdf

[39] W. Wang and Z. Lu, ‘‘Cyber security in smart grid: Survey and challenges,’’ Elsevier Comput. Netw., vol. 57, no. 5, pp. 1344–1371, Apr. 2013.

[40] C. Cortes and V. Vapnik, ‘‘Support-vector networks,’’ Mach. Learn., vol. 20, pp. 273–297, Sep. 1995, doi: 10.1023/A:1022627411411