

Detecting Web Attacks with End-To-End Deep Learning

By

Kanaka Durga Kantheti & Keerthi Surampalli

Approved By:

Main Advisor

Committee Members

Name:

Name:

Committee Member

Department Chair

Name:

Name:

Gannon University

Erie, Pennsylvania 16541

Spring, 2024

ABSTRACT

Web applications are popular targets for cyber-attacks because they are network-accessible and often contain vulnerabilities. An intrusion detection system monitors web applications and issues alerts when an attack attempt is detected. Existing implementations of intrusion detection systems usually extract features from network packets or string characteristics of input that are manually selected as relevant to attack analysis. Manually selecting features, however, is time-consuming and requires in-depth security domain knowledge. Moreover, large amounts of labeled legitimate and attack request data are needed by supervised learning algorithms to classify normal and abnormal behaviors, which is often expensive and impractical to obtain for production web applications. This paper provides three contributions to the study of autonomic intrusion detection systems. First, we evaluate the feasibility of an unsupervised/semi-supervised approach for web attack detection based on the Robust Software Modeling Tool (RSMT), which autonomically monitors and characterizes the runtime behavior of web applications. Second, we describe how RSMT trains a stacked denoising autoencoder to encode and reconstruct the call graph for end-to-end deep learning, where a low-dimensional representation of the raw features with unlabeled request data is used to recognize anomalies by computing the reconstruction error of the request data. Third, we analyze the results of empirically testing RSMT on both synthetic datasets and production applications with intentional vulnerabilities. Our results show that the proposed approach can efficiently and accurately detect attacks, including SQL injection, cross-site scripting, and deserialization, with minimal domain knowledge and little labeled training data.

Table Of Contents

ABSTRACT.....	2
CHAPTER 1: INTRODUCTION.....	8
1.0: EMERGING TRENDS AND CHALLENGES:	8
1.1: OBJECTIVE OF THE PROJECT:	11
CHAPTER 2: LITERATURE SURVEY:.....	13
2.0: A Classification of SQL-Injection Attacks and Countermeasures:.....	13
2.1: Static Detection of Cross-Site Scripting Vulnerabilities:.....	13
2.2: Defensive Programming:	14
2.3: Effects Of Cyber Security Knowledge on Attack Detection:	15
2.4: The Class Imbalance Problem:	15
2.5: A Hierarchical Intrusion Detection Model Based on The PCA Neural Networks:	16
2.6: An Adaptive Network Intrusion Detection Method Based on PCA And Support Vector Machines:	16
2.7: Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection:	17
2.8: ImageNet Classification with Deep Convolutional Neural Networks:	18
2.9: Sequence To Sequence Learning with Neural Networks:.....	18
CHAPTER 3: SYSTEM ANALYSIS	20
3.0: EXISTING SYSTEM:	20
3.1: PROPOSED SYSTEM:	20
3.2: PROCESS MODEL USED WITH JUSTIFICATION:	23
3.2.1: SDLC (Umbrella Model):	23
3.3: SOFTWARE REQUIREMENT SPECIFICATION:	32
3.3.0: Overall Description:.....	32
3.3.1: Feasibility.....	34
CHAPTER 4: SYSTEM DESIGN	36
4.0: UML DIAGRAM:	36
4.0.1: Class Diagram:.....	36
4.0.2: Use Case Diagram:	37
4.0.3: Sequence Diagram:	38
4.0.4: Collaboration Diagram:	40

4.0.5: Component Diagram:	41
4.0.6: Deployment Diagram:	42
4.0.7: Activity Diagram:	44
4.0.8: Data Flow Diagram:	45
CHAPTER 5: IMPLEMENTATION	48
5.0: PYTHON:	48
5.0.1: History Of Python:	48
5.0.1: Features Of Python:	49
5.0.1: Applications Of Python:	51
5.2: SAMPLE CODE:	52
CHAPTER 6: TESTING	52
6.0: IMPLEMENTATION AND TESTING:	58
6.0.1: System Testing:	59
6.0.2: Module Testing:	59
6.0.3: Integration Testing:	60
6.0.4: Acceptance Testing:	60
CHAPTER 7: SCREENSHOTS	65
CHAPTER 8: CONCLUSION	76
CHAPTER 9: REFERENCES	78

List of Figures

Figure 1: Requirements Gathering-----	24
Figure 2: Software Analysis -----	26
Figure 3: Software Designing -----	27
Figure 4: Software Development -----	29
Figure 5: Integration and Testing -----	30
Figure 6: Installation and Acceptance Test-----	31
Figure 7: Class Diagram -----	36
Figure 8: Use Case Diagram-----	38
Figure 9: Sequence Diagram -----	39
Figure 10: Collaboration Diagram-----	41
Figure 11: Component Diagram -----	42
Figure 12: Deployment Diagram-----	43
Figure 13: Activity Diagram-----	45
Figure 14: Data Flow Diagram-----	47
Figure 15: run.bat terminal-----	65
Figure 16: index.html page -----	66
Figure 17: User Signup Screen-----	67
Figure 18: User Login Screen -----	68
Figure 19: OTP validation screen-----	68
Figure 20: OTP Validation Screen 2 -----	69
Figure 21: Home Page-----	70
Figure 22: Upload Dataset Prompt -----	70

Figure 23: Dataset Upload & Processing Screen -----	71
Figure 24: Algorithm output Screen-----	72
Figure 25: Run Auto Encoder Output -----	73
Figure 26: Run Extension LSTM output-----	74
Figure 27: Output Graph -----	75

List of Tables

Table 1: Project Information**Error! Bookmark not defined.**

Table 2: Existing systems analysis**Error! Bookmark not defined.**

Table 3: Project team.....**Error! Bookmark not defined.**

Table 4: Acceptance Testing 60

CHAPTER 1: INTRODUCTION

1.0: EMERGING TRENDS AND CHALLENGES:

Emerging trends and challenges. Web applications are attractive targets for cyber attackers. SQL injection, cross site scripting (XSS) and remote code execution are common attacks that can disable web services, steal sensitive user information, and cause significant financial loss to both service providers and users. Protecting web applications from attack is hard. Even though developers and researchers have developed many counter-measures (such as firewalls, intrusion detection systems (IDSs) and defensive programming best practices) to protect web applications, web attacks remain a major threat. For example, researchers found that more than half of web applications during a 2015–2016 scan contained significant security vulnerabilities, such as XSS or SQL Injection. Moreover, hacking attacks cost the average American firm \$15.4 million per year. The Equifax data breach in 2017 (which exploited a vulnerability in Apache Struts) exposed over 143 million American consumers' sensitive personal information. Although the vulnerability was disclosed and patched in March 2017, Equifax took no action until four months later, which led to an estimated insured loss of over 125 million dollars. Conventional intrusion detection systems do not work as well as expected for a number of reasons, including the following:

- **Workforce limitations:** In-depth domain knowledge of web security is needed for web developers and network operators to deploy these systems. An experienced security expert is often needed to determine what features are relevant to extract from network packages, binaries, or other inputs for intrusion detection systems. Due to the large demand and relatively low barrier to entry into the software profession, however, many developers lack the necessary knowledge of secure coding practices.

- **Classification limitations:** Many intrusion detection systems rely on rule-based strategies or supervised machine learning algorithms to differentiate normal requests from attack requests, which requires large amounts of labeled training data to train the learning algorithms. It is hard and expensive, however, to obtain this training data for arbitrary custom applications. In addition, labeled training data is often heavily imbalanced since attack requests for custom systems are harder to get than normal requests, which poses challenges for classifiers. Moreover, although rule-based or supervised learning approaches can distinguish existing known attacks, new types of attacks and vulnerabilities emerge continuously, so they may be misclassified.

- **False positive limitations:** Although prior work has applied unsupervised learning algorithms (such as PCA and SVM) to detect web attacks, these approaches require manual selection of attack-specific features. Moreover, while these approaches achieve acceptable performance, they also incur false positive rates that are too high in practice, e.g., a 1% increase in false positives may cause an intrusion detection system to incorrectly flag thousands of legitimate users. It is therefore essential to reduce the false positive rate of these systems. Given these challenges with using conventional intrusion detection systems, an infrastructure that requires less expertise and labeled training data is needed.

Solution approach \Rightarrow Applying end-to-end deep learning to detect cyber-attacks autonomically in real-time and adapt efficiently, scalable, and securely to thwart them. This paper explores the potential of end-to-end deep learning in intrusion detection systems. Our approach applies deep learning to the entire process from feature engineering to prediction, i.e., raw input is fed into the network and high-level output is generated directly. There is thus no need for users to select features and construct large labeled training sets manually. We empirically evaluate how well an unsupervised- /semi-supervised learning approach based on

end-to-end deep learning detects web attacks. Our work is motivated by the success deep learning has achieved in computer vision, speech recognition, and natural language processing. In particular, deep learning is not only capable of classification, but also automatically extracting features from high dimensional raw input. Our deep learning approach is based on the Robust Software Modeling Tool (RSMT), which is a late-stage (i.e., post-compilation) instrumentation-based tool chain that target languages designed to run on the Java Virtual Machine (JVM). RSMT is a general-purpose tool that extracts arbitrarily fine-grained traces of program execution from running software, which is applied in this paper to detect intrusions at runtime by extracting call traces in web applications. Our approach applies RSMT in the following steps:

1. During an unsupervised training epoch, traces generated by test suites are used to learn a model of correct program execution with a stacked denoising autoencoder, which is a symmetric deep neural network trained to have target value equal to a given input value
2. A small amount of labeled data is then used to calculate reconstruction error and establish a threshold to distinguish normal and abnormal behaviors.
3. During a subsequent validation epoch, traces extracted from a live application are classified using previously learned models to determine whether each trace is indicative of normal or abnormal behavior.

A key contribution of this paper is the integration of autonomic runtime behavior monitoring and characterization of web applications with end-to-end deep learning mechanisms, which generate high-level output directly from raw feature input.

This paper extends our prior work by focusing on attack detection using stacked denoising auto encoders. This improved approach significantly improves upon our past approaches that

relied on other machine learning techniques and typically required labeled training sets. A key benefit of the approaches presented in this paper versus our prior work is that they do not require manual feature engineering, which is needed for our past detection techniques. Moreover, the approaches work well with standard software engineering artifacts, the execution data from tests, which can be gleaned from many application-types. The remainder of this paper is organized as follows: Section 2 summarizes the key research challenges we are addressing in our work; Section 3 describes the structure and functionality of the Robust Software Modeling Tool (RSMT); Section 4 explains our approach for web attack detection using unsupervised/semi-supervised end-to-end deep learning and the stacked denoising autoencoder; Section 5 empirically evaluates the performance of our RSMT-based intrusion detection system on representative web applications; Section 6 compares our work with related web attack detection techniques; and Section 7 presents concluding remarks.

1.1: OBJECTIVE OF THE PROJECT:

Web applications are popular targets for cyber-attacks because they are network-accessible and often contain vulnerabilities. An intrusion detection system monitors web applications and issues alerts when an attack attempt is detected. Existing implementations of intrusion detection systems usually extract features from network packets or string characteristics of input that are manually selected as relevant to attack analysis. Manually selecting features, however, is time-consuming and requires in-depth security domain knowledge. Moreover, large amounts of labeled legitimate and attack request data are needed by supervised learning algorithms to classify normal and abnormal behaviors, which is often expensive and impractical to obtain for production web applications. This paper provides three contributions to the study of autonomic

intrusion detection systems. First, we evaluate the feasibility of an unsupervised/semi-supervised approach for web attack detection based on the Robust Software Modeling Tool (RSMT), which autonomically monitors and characterizes the runtime behavior of web applications. Second, we describe how RSMT trains a stacked denoising autoencoder to encode and reconstruct the call graph for end-to-end deep learning, where a low-dimensional representation of the raw features with unlabeled request data is used to recognize anomalies by computing the reconstruction error of the request data. Third, we analyze the results of empirically testing RSMT on both synthetic datasets and production applications with intentional vulnerabilities. Our results show that the proposed approach can efficiently and accurately detect attacks, including SQL injection, cross-site scripting, and deserialization, with minimal domain knowledge and little labeled training data.

CHAPTER 2: LITERATURE SURVEY:

2.0: A Classification of SQL-Injection Attacks and Countermeasures:

SQL injection attacks pose a serious security threat to Web applications: they allow attackers to obtain unrestricted access to the databases underlying the applications and to the potentially sensitive information these databases contain. Although researchers and practitioners have proposed various methods to address the SQL injection problem, current approaches either fail to address the full scope of the problem or have limitations that prevent their use and adoption. Many researchers and practitioners are familiar with only a subset of the wide range of techniques available to attackers who are trying to take advantage of SQL injection vulnerabilities. As a consequence, many solutions proposed in the literature address only some of the issues related to SQL injection. To address this problem, we present an extensive review of the different types of SQL injection attacks known to date. For each type of attack, we provide descriptions and examples of how attacks of that type could be performed. We also present and analyze existing detection and prevention techniques against SQL injection attacks. For each technique, we discuss its strengths and weaknesses in addressing the entire range of SQL injection attacks.

2.1: Static Detection of Cross-Site Scripting Vulnerabilities:

Web applications support many of our daily activities, but they often have security problems, and their accessibility makes them easy to exploit. In cross-site scripting (XSS), an attacker exploits the trust a Web client (browser) has for a trusted server and executes injected script on the browser with the server's privileges. In 2006, XSS constituted the largest class of newly reported vulnerabilities making it the most prevalent class of attacks today. Web applications have XSS vulnerabilities because the validation they perform on untrusted input

does not suffice to prevent that input from invoking a browser's JavaScript interpreter, and this validation is particularly difficult to get right if it must admit some HTML mark-up. Most existing approaches to finding XSS vulnerabilities are taint-based and assume input validation functions to be adequate, so they either miss real vulnerabilities or report many false positives. This paper presents a static analysis for finding XSS vulnerabilities that directly addresses weak or absent input validation. Our approach combines work on tainted information flow with string analysis. Proper input validation is difficult largely because of the many ways to invoke the JavaScript interpreter; we face the same obstacle checking for vulnerabilities statically, and we address it by formalizing a policy based on the W3C recommendation, the Firefox source code, and online tutorials about closed-source browsers. We provide effective checking algorithms based on our policy. We implement our approach and provide an extensive evaluation that finds both known and unknown vulnerabilities in real-world web applications.

2.2: Defensive Programming:

Using an annotation toolkit to build dos-resistant software

This paper describes a toolkit to help improve the robustness of code against DoS attacks. We observe that when developing software, programmers primarily focus on functionality. Protecting code from attacks is often considered the responsibility of the OS, firewalls and intrusion detection systems. As a result, many DoS vulnerabilities are not discovered until the system is attacked and the damage is done. Instead of reacting to attacks after the fact, this paper argues that a better solution is to make software defensive by systematically injecting protection mechanisms into the code itself. Our toolkit provides an API that programmers use to annotate their code. At runtime, these annotations serve as both sensors and actuators: watching for resource abuse and taking the appropriate action should abuse be detected. This paper presents

the design and implementation of the toolkit, as well as evaluation of its effectiveness with three widely-deployed network services.

2.3: Effects Of Cyber Security Knowledge on Attack Detection:

Ensuring cyber security is a complex task that relies on domain knowledge and requires cognitive abilities to determine possible threats from large amounts of network data. This study investigates how knowledge in network operations and information security influence the detection of intrusions in a simple network. We developed a simplified Intrusion Detection System (IDS), which allows us to examine how individuals with or without knowledge in cyber security detect malicious events and declare an attack based on a sequence of network events. Our results indicate that more knowledge in cyber security facilitated the correct detection of malicious events and decreased the false classification of benign events as malicious. However, knowledge had less contribution when judging whether a sequence of events representing a cyber-attack. While knowledge of cyber security helps in the detection of malicious events, situated knowledge regarding a specific network at hand is needed to make accurate detection decisions. Responses from participants that have knowledge in cyber security indicated that they were able to distinguish between different types of cyber-attacks, whereas novice participants were not sensitive to the attack types. We explain how these findings relate to cognitive processes and we discuss their implications for improving cyber security.

2.4: The Class Imbalance Problem:

A systematic study

In machine learning problems, differences in prior class probabilities -- or class imbalances -- have been reported to hinder the performance of some standard classifiers, such as decision

trees. This paper presents a systematic study aimed at answering three different questions. First, we attempt to understand the nature of the class imbalance problem by establishing a relationship between concept complexity, size of the training set and class imbalance level. Second, we discuss several basic re-sampling or cost-modifying methods previously proposed to deal with the class imbalance problem and compare their effectiveness. The results obtained by such methods on artificial domains are linked to results in real-world domains. Finally, we investigate the assumption that the class imbalance problem does not only affect decision tree systems but also affects other classification systems such as Neural Networks and Support Vector Machines.

2.5: A Hierarchical Intrusion Detection Model Based on The PCA Neural Networks:

Most of existing intrusion detection (ID) models with a single-level structure can only detect either misuse or anomaly attacks. A hierarchical ID model using principal component analysis (PCA) neural networks is proposed to overcome such shortages. In the proposed model, PCA is applied for classification and neural networks are used for online computing. Experimental results and comparative studies based on the 1998 DARPA evaluation data sets are given, which show the proposed model can classify the network connections with satisfying performance.

2.6: An Adaptive Network Intrusion Detection Method Based on PCA And Support Vector Machines:

Network intrusion detection is an important technique in computer security. However, the performance of existing intrusion detection systems (IDSs) is unsatisfactory since new attacks are constantly developed and the speed of network traffic volumes increases fast. To improve the performance of IDSs both in accuracy and speed, this paper proposes a novel adaptive intrusion detection method based on principal component analysis (PCA) and support vector machines

(SVMs). By making use of PCA, the dimension of network data patterns is reduced significantly. The multi-class SVMs are employed to construct classification models based on training data processed by PCA. Due to the generalization ability of SVMs, the proposed method has good classification performance without tedious parameter tuning. Dimension reduction using PCA may improve accuracy further. The method is also superior to SVMs without PCA in fast training and detection speed. Experimental results on KDD-Cup99 intrusion detection data illustrate the effectiveness of the proposed method.

2.7: Using Adaptive Alert Classification to Reduce False Positives in Intrusion Detection:

Intrusion Detection Systems (IDSs) are used to monitor computer systems for signs of security violations. Having detected such signs, IDSs trigger alerts to report them. These alerts are presented to a human analyst, who evaluates them and initiates an adequate response. In practice, IDSs have been observed to trigger thousands of alerts per day, most of which are false positives (i.e., alerts mistakenly triggered by benign events). This makes it extremely difficult for the analyst to correctly identify the true positives (i.e., alerts related to attacks). In this paper we describe ALAC, the Adaptive Learner for Alert Classification, which is a novel system for reducing false positives in intrusion detection. The system supports the human analyst by classifying alerts into true positives and false positives. The knowledge of how to classify alerts is learned adaptively by observing the analyst. Moreover, ALAC can be configured to process autonomously alerts that have been classified with high confidence. For example, ALAC may discard alerts that were classified with high confidence as false positive. That way, ALAC effectively reduces the analyst's workload. We describe a prototype implementation of ALAC

and the choice of a suitable machine learning technique. Moreover, we experimentally validate ALAC and show how it facilitates the analyst's work.

2.8: ImageNet Classification with Deep Convolutional Neural Networks:

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce over fitting in the fully-connected layers we employed a recently-developed regularization method called “dropout” that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

2.9: Sequence To Sequence Learning with Neural Networks:

Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. In this paper, we present a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Our main result is that on an English to French translation task from the WMT'14 dataset, the translations produced by the LSTM achieve a

BLEU score of 34.8 on the entire test set, where the LSTM's BLEU score was penalized on out-of-vocabulary words. Additionally, the LSTM did not have difficulty on long sentences. For comparison, a phrase-based SMT system achieves a BLEU score of 33.3 on the same dataset. When we used the LSTM to rerank the 1000 hypotheses produced by the aforementioned SMT system, its BLEU score increases to 36.5, which is close to the previous best result on this task. The LSTM also learned sensible phrase and sentence representations that are sensitive to word order and are relatively invariant to the active and the passive voice. Finally, we found that reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly, because doing so introduced many short-term dependencies between the source and the target sentence which made the optimization problem easier.

CHAPTER 3: SYSTEM ANALYSIS

3.0: EXISTING SYSTEM:

Web applications are popular targets for cyber-attacks because they are network-accessible and often contain vulnerabilities. An intrusion detection system monitors web applications and issues alerts when an attack attempt is detected. Existing implementations of intrusion detection systems usually extract features from network packets or string characteristics of input that are manually selected as relevant to attack analysis. Manually selecting features, however, is time-consuming and requires in-depth security domain knowledge. Moreover, large amounts of labeled legitimate and attack request data are needed by supervised learning algorithms to classify normal and abnormal behaviors, which is often expensive and impractical to obtain for production web applications.

Disadvantage:

1. Less Security.

3.1: PROPOSED SYSTEM:

This paper provides three contributions to the study of autonomic intrusion detection systems. First, we evaluate the feasibility of an unsupervised/semi-supervised approach for web attack detection based on the Robust Software Modeling Tool (RSMT), which autonomically monitors and characterizes the runtime behavior of web applications. Second, we describe how RSMT trains a stacked denoising autoencoder to encode and reconstruct the call graph for end-to-end deep learning, where a low-dimensional representation of the raw features with unlabeled request data is used to recognize anomalies by computing the reconstruction error of the request data. Third, we analyze the results of empirically testing RSMT on both synthetic datasets and

production applications with intentional vulnerabilities. Our results show that the proposed approach can efficiently and accurately detect attacks, including SQL injection, cross-site scripting, and deserialization, with minimal domain knowledge and little labeled training data.

Advantage:

1. More Security.

Modules:

1. Upload RSMT Traces Dataset
2. Generate Train & Test Model
3. Run SVM Algorithm
4. Run Naive Bayes algorithm
5. Run Propose AutoEncoder Deep Learning Algorithm
6. Run Extension LSTM Algorithm
7. Precision comparison Graph
8. Recall Comparison Graph
9. FScore Comparison Graph

Upload RSMT Traces Dataset

Upload RSMT Traces Dataset is the first module of the our project and it is used to upload attack traces.

Generate Train & Test Model

Generate Train & Test Model module is used to generate train and test data. All deep learning algorithms will take 80% dataset as training and 20% dataset for testing.

Run SVM Algorithm

Run SVM Algorithm module is used to generate SVM model on train data and calculate precision, recall on test data.

Run Naive Bayes algorithm

Run Naive Bayes Algorithm module is used to generate Naive Bayes model on train data and calculate precision, recall on test data.

Run Propose AutoEncoder Deep Learning Algorithm

Run Propose Auto Encoder module to run propose algorithm, Auto Encoder got 90% accuracy.

Run Extension LSTM Algorithm

Run Extension LSTM Algorithm module is used to run LSTM and we got values for LSTM algorithm.

Precision comparison Graph

Precision Comparison graph x-axis represents algorithm name and y-axis represents precision value. In all algorithm propose Auto Encoder showing good performance.

Recall Comparison Graph

Recall Comparison graph x-axis represents algorithm name and y-axis represents recall value. In all algorithm Extension LSTM showing good performance.

FScore Comparison Graph

FScore comparison graph x-axis represents algorithm name and y-axis represents FScore value. In all algorithm Extension LSTM showing good performance.

3.2: PROCESS MODEL USED WITH JUSTIFICATION:

3.2.1: SDLC (Umbrella Model):

Stages in SDLC:

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

Requirements Gathering stage:

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.

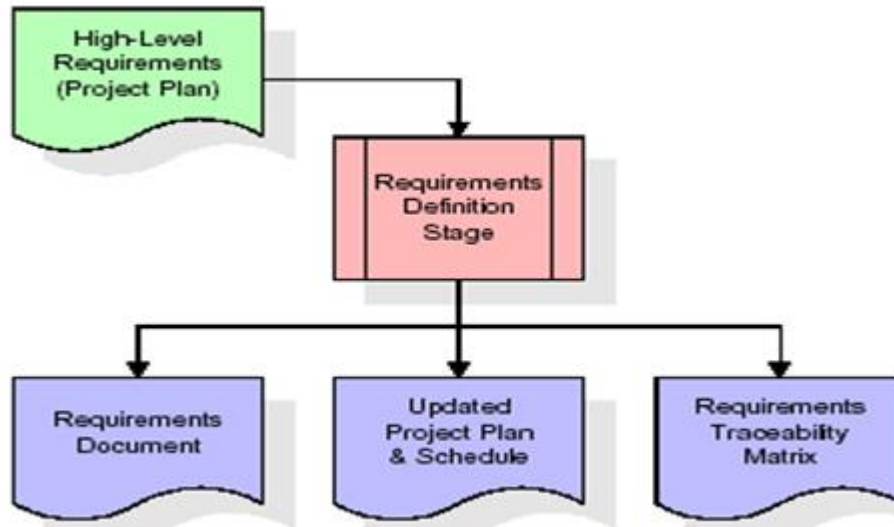


Figure 1: Requirements Gathering

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. In this format, each requirement can be traced to a specific product goal, hence the term requirements traceability.

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.

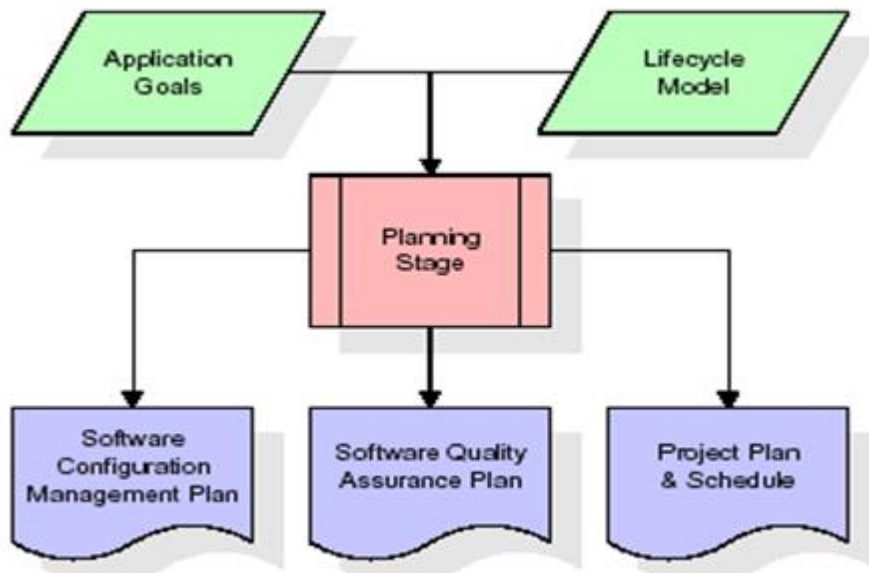


Figure 2: Software Analysis

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.

Designing Stage:

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.

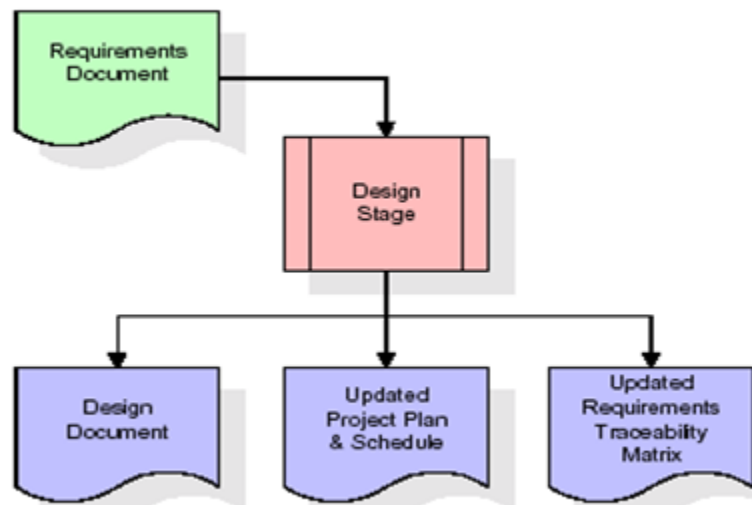


Figure 3: Software Designing

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.

Development (Coding) Stage:

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, and 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.

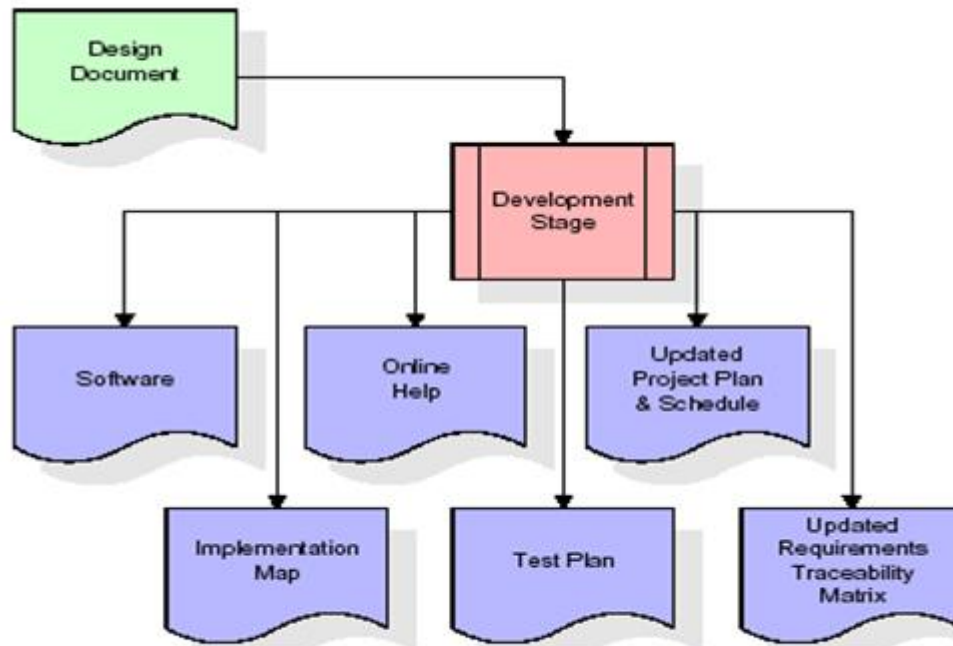


Figure 4: Software Development

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.

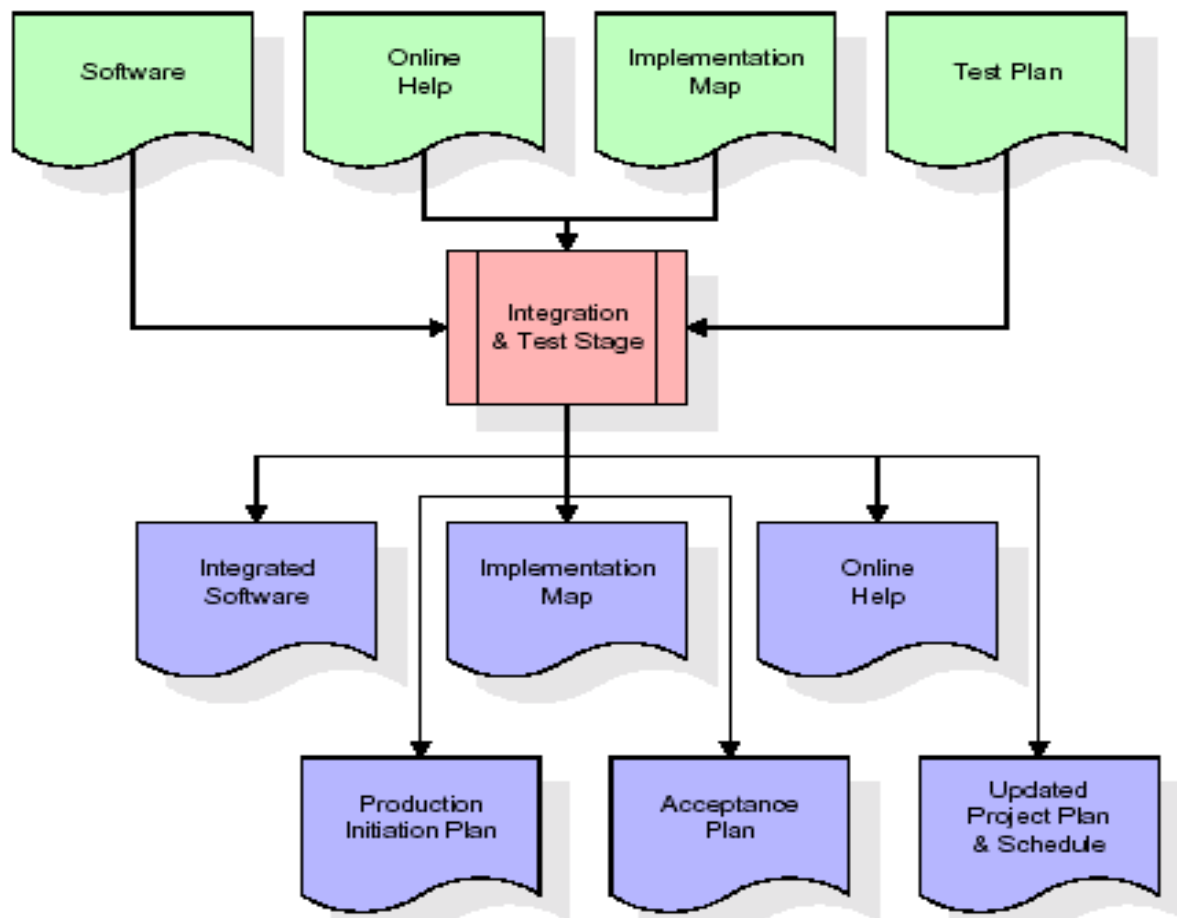


Figure 5: Integration and Testing

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.

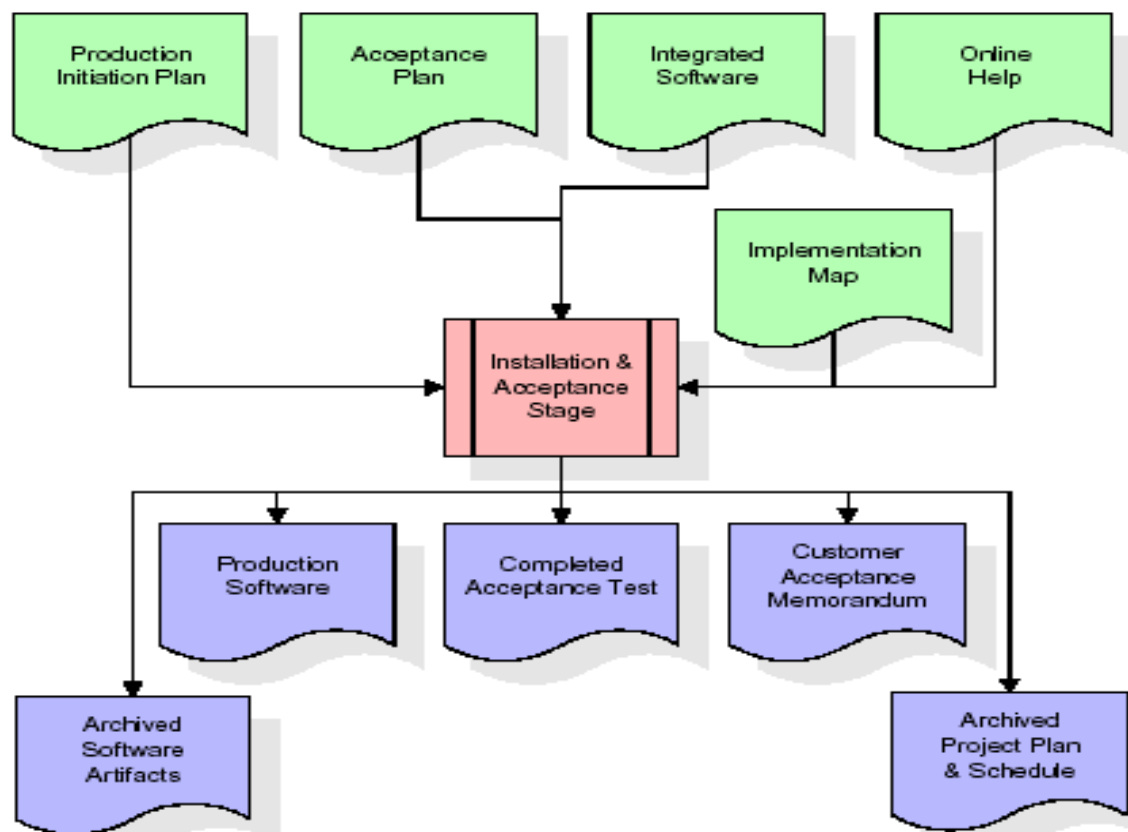


Figure 6: Installation and Acceptance Test

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 labor 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. For this life cycle there is no end, it will be continued so on like an umbrella (no ending point to umbrella sticks).

3.3: SOFTWARE REQUIREMENT SPECIFICATION:

3.3.0: Overall Description:

A Software Requirements Specification (SRS) – a requirements specification for a software system is a complete description of the behavior of a system to be developed. It includes a set of use cases that describe all the interactions the users will have with the software. In addition to use cases, the SRS also contains non-functional requirements. Nonfunctional requirements are requirements which impose constraints on the design or implementation (such as performance engineering requirements, quality standards, or design constraints).

System requirements specification: A structured collection of information that embodies the requirements of a system. A business analyst, sometimes titled system analyst, is responsible for analyzing the business needs of their clients and stakeholders to help identify business problems and propose solutions. Within the systems development lifecycle domain, the BA typically performs a liaison function between the business side of an enterprise and the information technology department or external service providers. Projects are subject to three sorts of requirements:

- Business requirements describe in business terms *what* must be delivered or accomplished to provide value.
- Product requirements describe properties of a system or product (which could be one of several ways to accomplish a set of business requirements.)

Process requirements describe activities performed by the developing organization. For instance, process requirements could specify. Preliminary investigation examines project feasibility, the likelihood the system will be useful to the organization. The main objective of the feasibility study is to test the Technical, Operational and Economical feasibility for adding new modules and debugging old running system. All system is feasible if they are unlimited resources and infinite time. There are aspects in the feasibility study portion of the preliminary investigation:

3.3.1: Feasibility

Economic Feasibility

A system can be developed technically and that will be used if installed must still be a good investment for the organization. In the economic feasibility, the development cost in creating the system is evaluated against the ultimate benefit derived from the new systems. Financial benefits must equal or exceed the costs. The system is economically feasible. It does not require any addition hardware or software. Since the interface for this system is developed using the existing resources and technologies available at NIC, there is nominal expenditure and economic feasibility for certain.

Operational Feasibility

Proposed projects are beneficial only if they can be turned out into information system. That will meet the organization's operating requirements. Operational feasibility aspects of the project are to be taken as an important part of the project implementation. This system is targeted to be in accordance with the above-mentioned issues. Beforehand, the management issues and user requirements have been taken into consideration. So there is no question of resistance from the users that can undermine the possible application benefits. The well-planned design would

ensure the optimal utilization of the computer resources and would help in the improvement of performance status.

Technical Feasibility

Earlier no system existed to cater to the needs of 'Secure Infrastructure Implementation System'. The current system developed is technically feasible. It is a web-based user interface

for audit workflow at NIC-CSD. Thus it provides an easy access to .the users. The database's purpose is to create, establish and maintain a workflow among various entities in order to facilitate all concerned users in their various capacities or roles. Permission to the users would be granted based on the roles specified. Therefore, it provides the technical guarantee of accuracy, reliability and security.

CHAPTER 4: SYSTEM DESIGN

4.0: UML DIAGRAM:

4.0.1: Class Diagram:

The class diagram is the main building block of object oriented modeling. It is used both for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed. In the diagram, classes are represented with boxes which contain three parts:

- The upper part holds the name of the class
- The middle part contains the attributes of the class
- The bottom part gives the methods or operations the class can take or undertake.

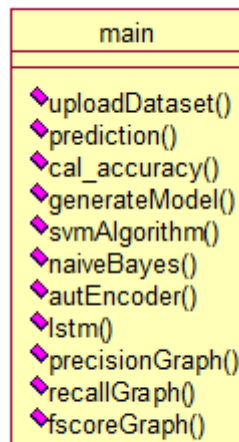


Figure 7: Class Diagram

4.0.2: Use Case Diagram:

A use case diagram at its simplest is a representation of a user's interaction with the system and depicting the specifications of a use case. A use case diagram can portray the different types of users of a system and the various ways that they interact with the system. This type of diagram is typically used in conjunction with the textual use case and will often be accompanied by other types of diagrams as well.

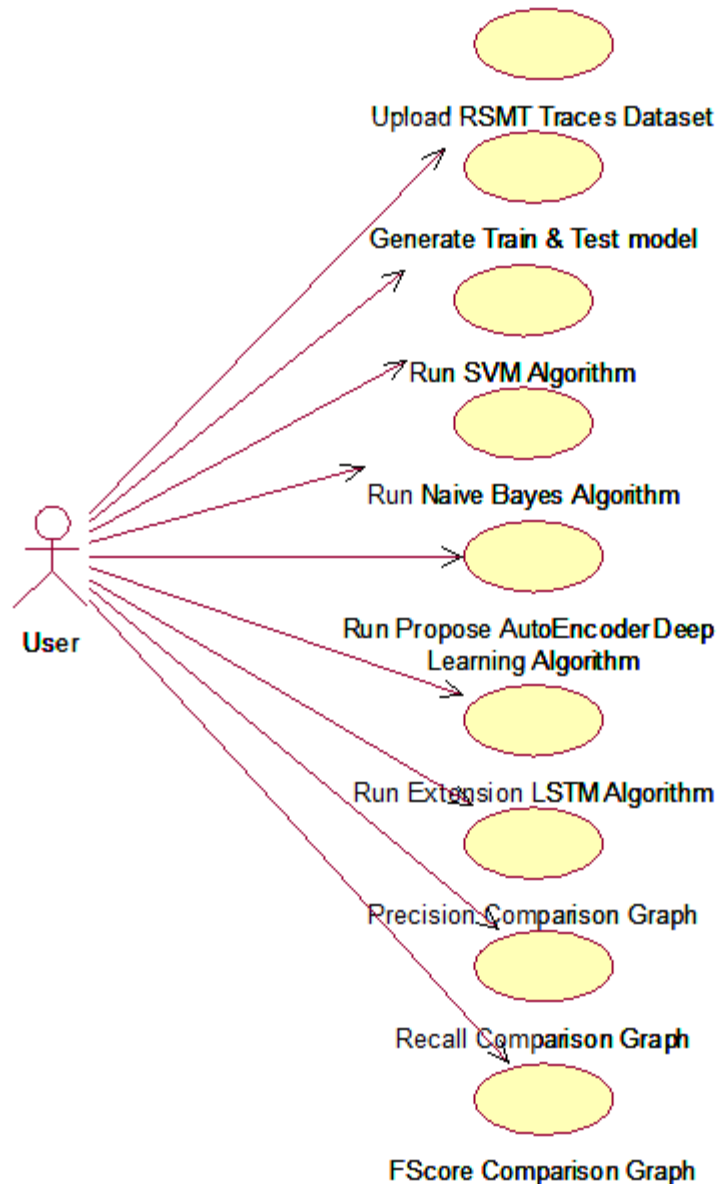


Figure 8: Use Case Diagram

4.0.3: Sequence Diagram:

A sequence diagram 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. A sequence

diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario. Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

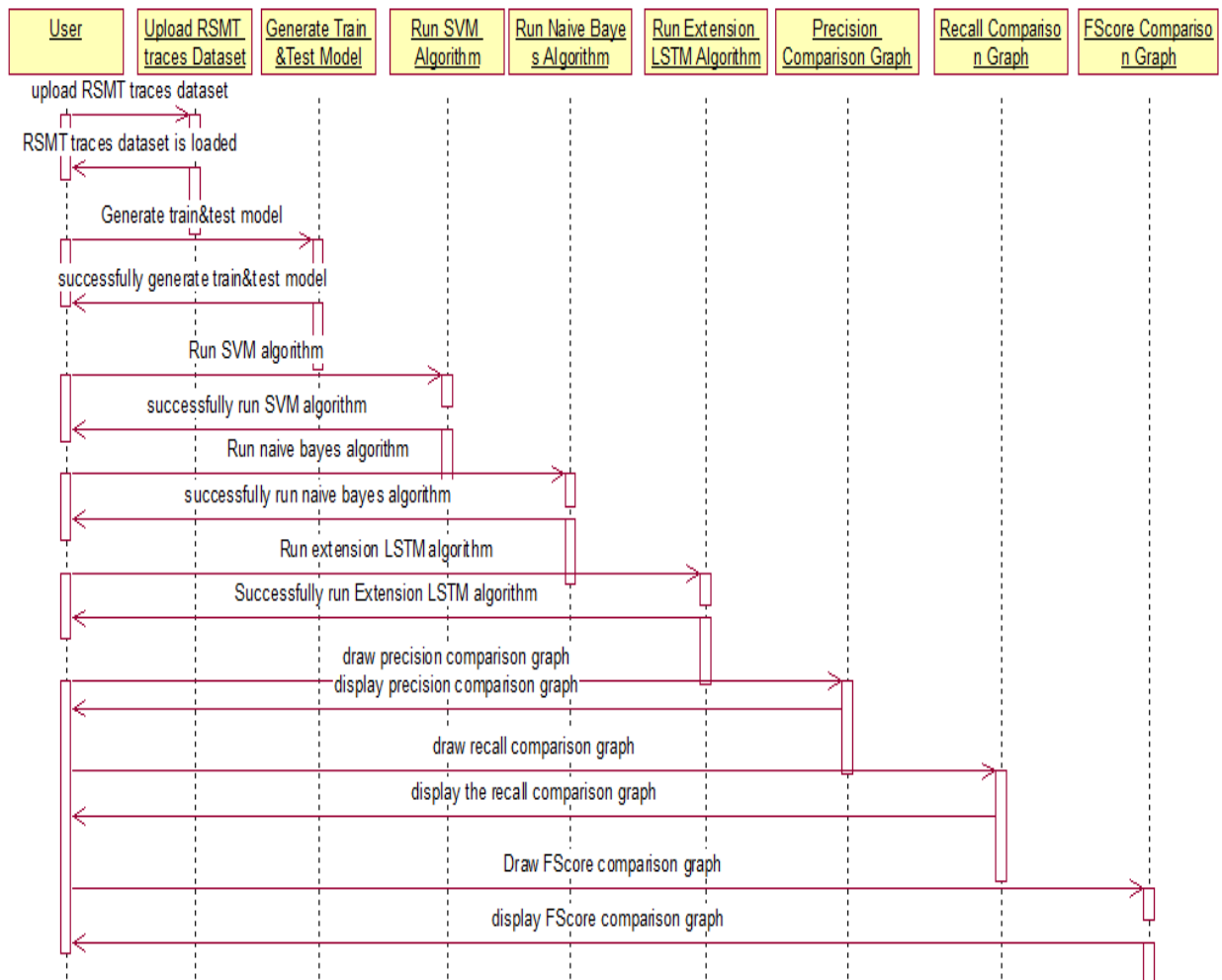


Figure 9: Sequence Diagram

4.0.4: Collaboration Diagram:

A collaboration diagram describes interactions among objects in terms of sequenced messages. Collaboration diagrams represent a combination of information taken from class, sequence, and use case diagrams describing both the static structure and dynamic behavior of a system.

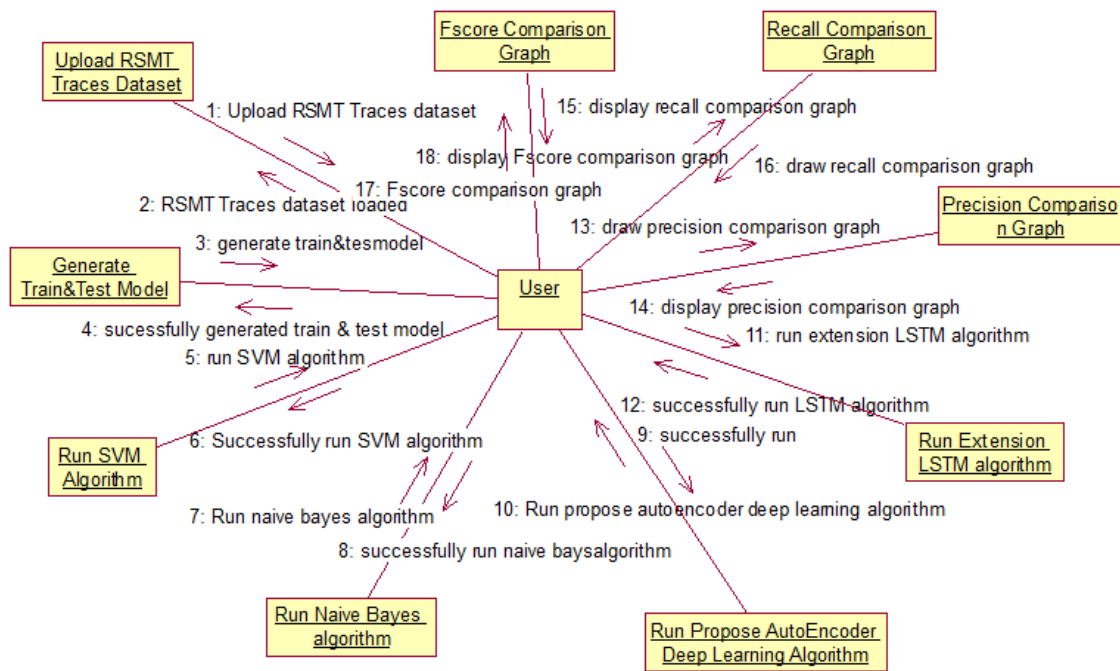


Figure 10: Collaboration Diagram

4.0.5: Component Diagram:

In the Unified Modeling Language, a component diagram depicts how components are wired together to form larger components and or software systems. They are used to illustrate the structure of arbitrarily complex systems.

Components are wired together by using an assembly connector to connect the required interface of one component with the provided interface of another component. This illustrates the service consumer - service provider relationship between the two components.

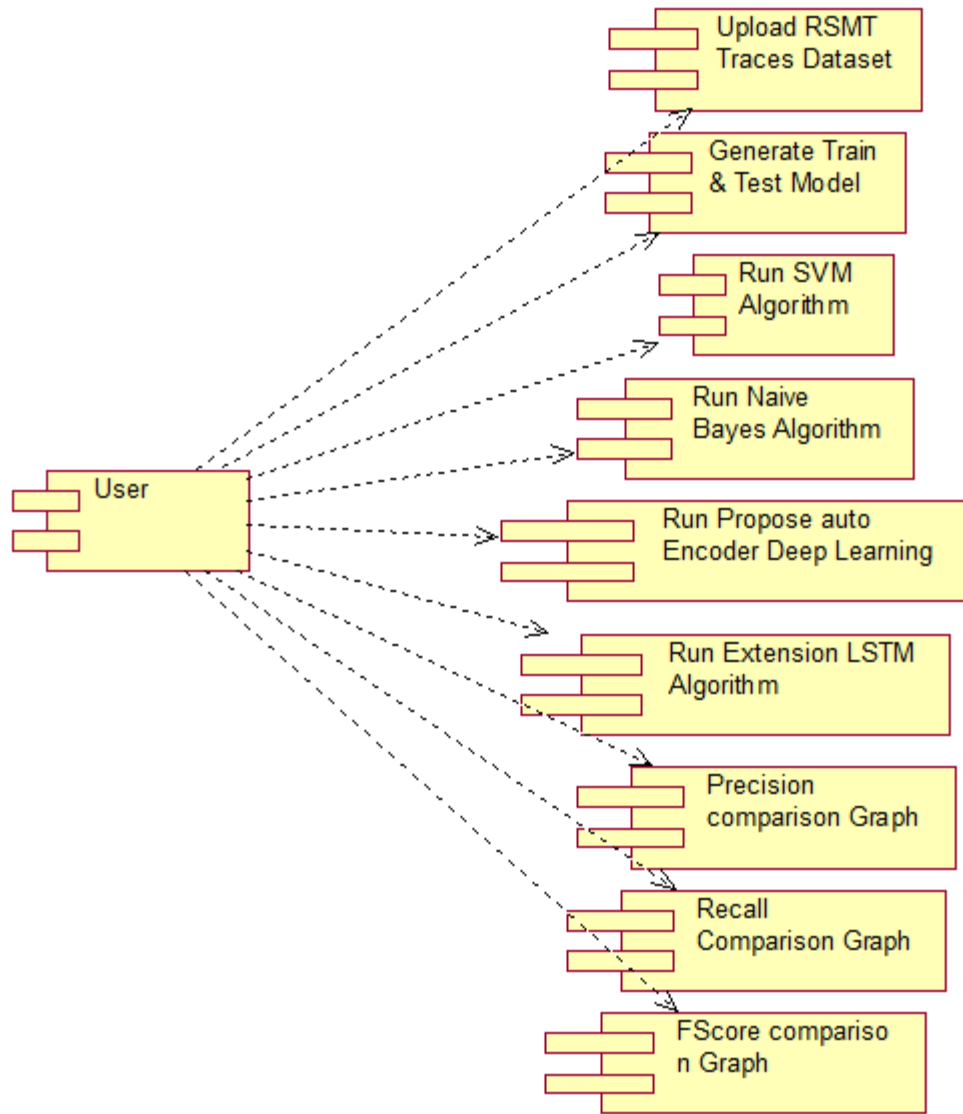


Figure 11: Component Diagram

4.0.6: Deployment Diagram:

A deployment diagram in the Unified Modeling Language models the *physical* deployment of artifacts on nodes. To describe a web site, for example, a deployment diagram would show what hardware components ("nodes") exist (e.g., a web server, an application server, and a

database server), what software components ("artifacts") run on each node (e.g., web application, database), and how the different pieces are connected (e.g. JDBC, REST, RMI).

The nodes appear as boxes, and the artifacts allocated to each node appear as rectangles within the boxes. Nodes may have sub nodes, which appear as nested boxes. A single node in a deployment diagram may conceptually represent multiple physical nodes, such as a cluster of database servers.

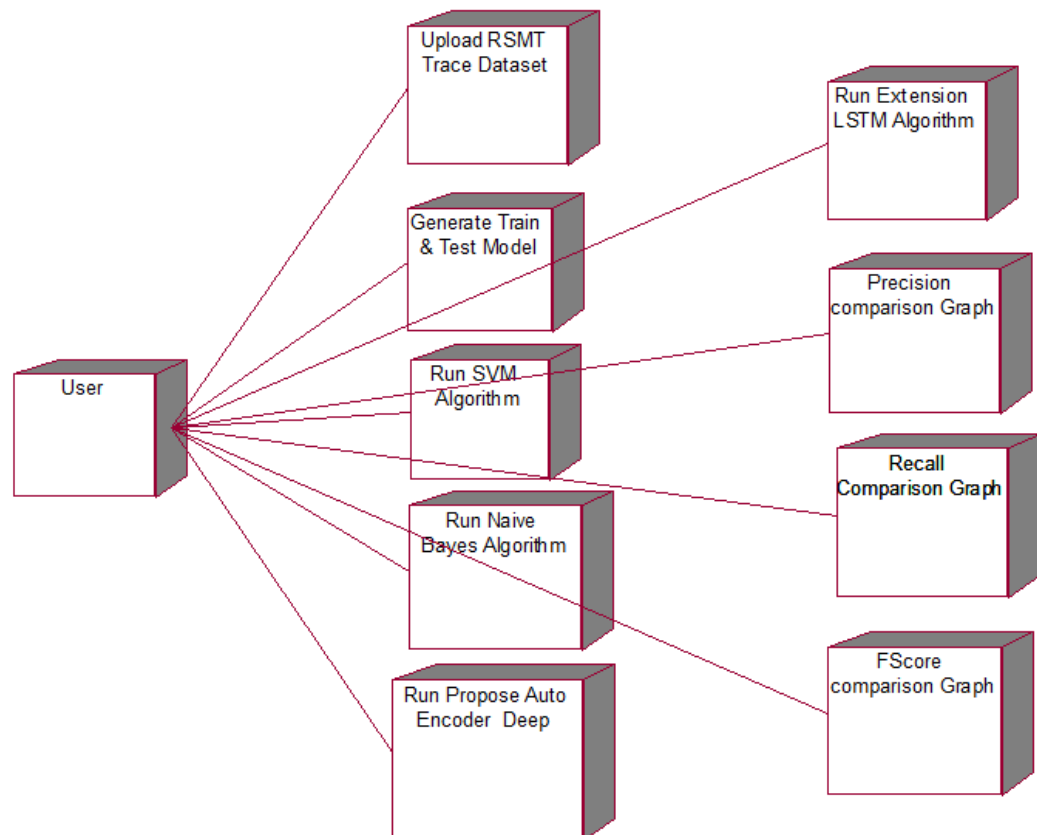


Figure 12: Deployment Diagram

4.0.7: Activity Diagram:

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. It is basically a flow chart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent.

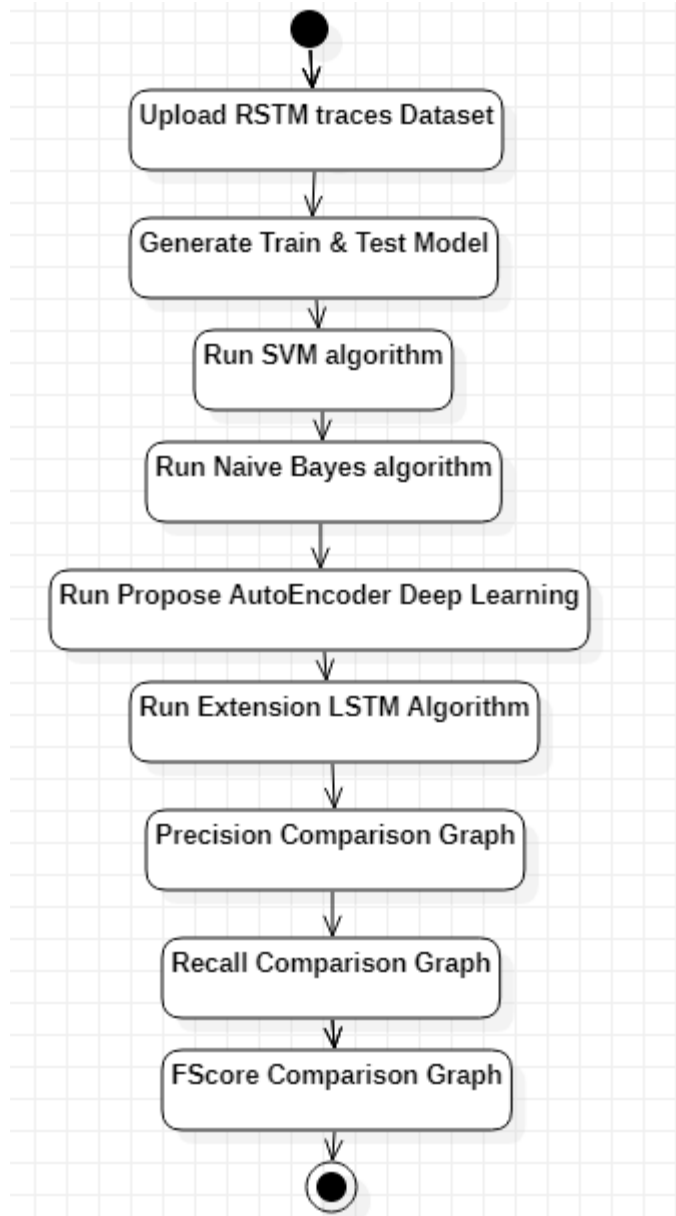


Figure 13: Activity Diagram

4.0.8: Data Flow Diagram:

Data flow diagrams illustrate how data is processed by a system in terms of inputs and outputs. Data flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analyzing

each of the functional areas of interest. This analysis can be carried out in precisely the level of detail required. The technique exploits a method called top-down expansion to conduct the analysis in a targeted way.

As the name suggests, Data Flow Diagram (DFD) is an illustration that explicates the passage of information in a process. A DFD can be easily drawn using simple symbols. Additionally, complicated processes can be easily automated by creating DFDs using easy-to-use, free downloadable diagramming tools. A DFD is a model for constructing and analyzing information processes. DFD illustrates the flow of information in a process depending upon the inputs and outputs. A DFD can also be referred to as a Process Model. A DFD demonstrates business or technical process with the support of the outside data saved, plus the data flowing from the process to another and the end results.

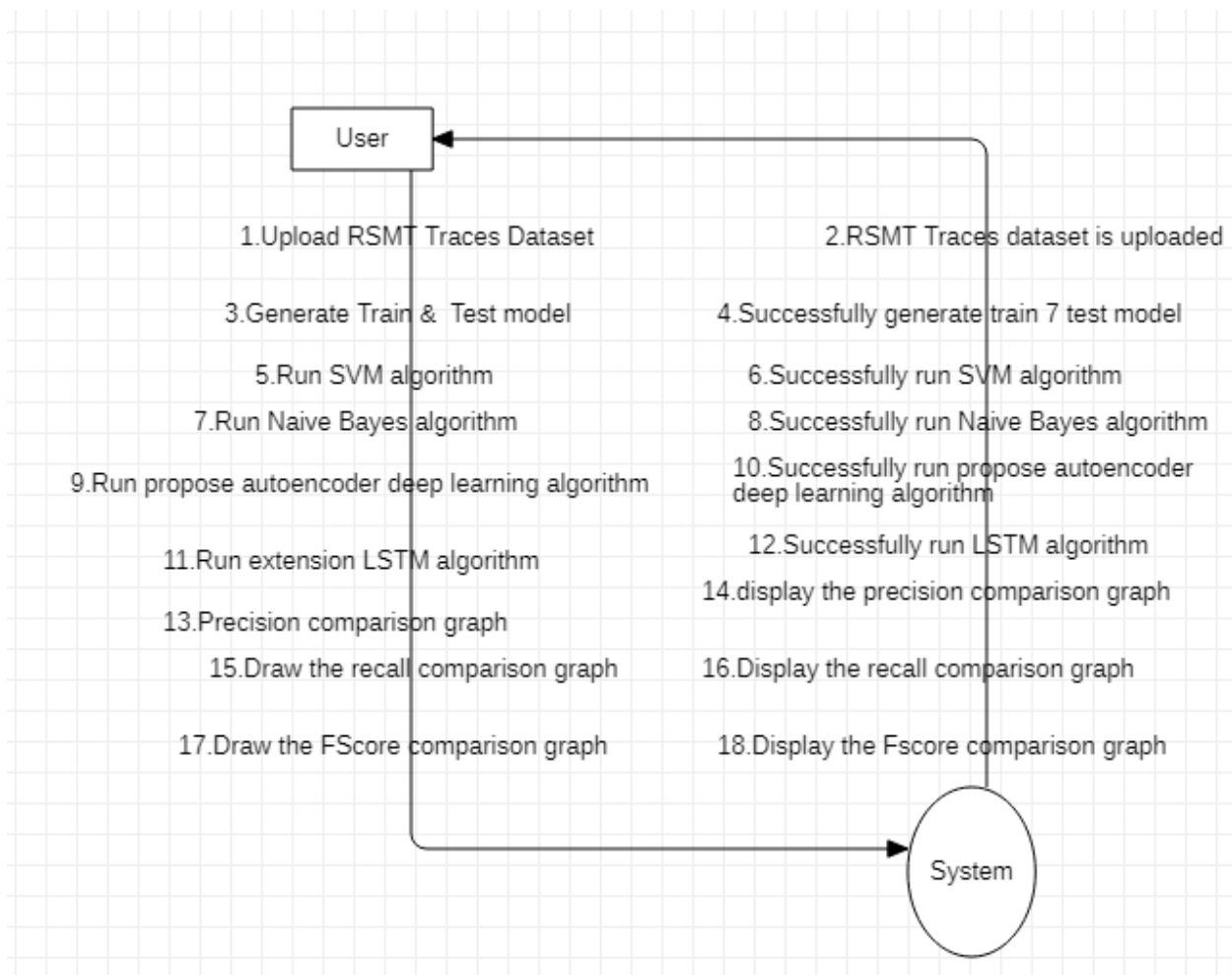


Figure 14: Data Flow Diagram

CHAPTER 5: IMPLEMENTATION

5.0: PYTHON:

Python is a general-purpose language. It has wide range of applications from Web development (like: Django and Bottle), scientific and mathematical computing (Orange, SymPy, NumPy) to desktop graphical user Interfaces (Pygame, Panda3D). The syntax of the language is clean and length of the code is relatively short. It's fun to work in Python because it allows you to think about the problem rather than focusing on the syntax.

5.0.1: History Of Python:

Python is a fairly old language created by Guido Van Rossum. The design began in the late 1980s and was first released in February 1991.

Why Python was created?

In late 1980s, Guido Van Rossum was working on the Amoeba distributed operating system group. He wanted to use an interpreted language like ABC (ABC has simple easy-to-understand syntax) that could access the Amoeba system calls. So, he decided to create a language that was extensible. This led to design of a new language which was later named Python.

Why the name Python?

No. It wasn't named after a dangerous snake. Rossum was fan of a comedy series from late seventies. The name "Python" was adopted from the same series "Monty Python's Flying Circus".

5.0.1: Features Of Python:

A simple language which is easier to learn

Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like: C++, Java, C#. Python makes programming fun and allows you to focus on the solution rather than syntax.

If you are a newbie, it's a great choice to start your journey with Python.

Free and open-source

You can freely use and distribute Python, even for commercial use. Not only can you use and distribute software written in it, you can even make changes to the Python's source code.

Python has a large community constantly improving it in each iteration.

Portability

You can move Python programs from one platform to another, and run it without any changes.

It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.

Extensible and Embeddable

Suppose an application requires high performance. You can easily combine pieces of C/C++ or other languages with Python code.

This will give your application high performance as well as scripting capabilities which other languages may not provide out of the box.

A high-level, interpreted language

Unlike C/C++, you don't have to worry about daunting tasks like memory management, garbage collection and so on.

Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You don't need to worry about any lower-level operations.

Large standard libraries to solve common tasks

Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself. For example: Need to connect MySQL database on a Web server? You can use MySQL dB library using `import MySQL dB` .

Standard libraries in Python are well tested and used by hundreds of people. So, you can be sure that it won't break your application.

Object-oriented

Everything in Python is an object. Object oriented programming (OOP) helps you solve a complex problem intuitively.

With OOP, you are able to divide these complex problems into smaller sets by creating objects.

5.0.1: Applications Of Python:

1. Simple Elegant Syntax

Programming in Python is fun. It's easier to understand and write Python code. Why? The syntax feels natural. Take this source code for an example:

```
a = 2
```

```
b = 3
```

```
sum = a + b
```

```
print(sum)
```

2. Not overly strict

You don't need to define the type of a variable in Python. Also, it's not necessary to add semicolon at the end of the statement.

Python enforces you to follow good practices (like proper indentation). These small things can make learning much easier for beginners.

3. Expressiveness of the language

Python allows you to write programs having greater functionality with fewer lines of code. Here's a link to the source code of Tic-tac-toe game with a graphical interface and a smart computer opponent in less than 500 lines of code. This is just an example. You will be amazed how much you can do with Python once you learn the basics.

4. Great Community and Support

Python has a large supporting community. There are numerous active forums online which can be handy if you are stuck.

5.2: SAMPLE CODE:

```
from tkinter import messagebox
from tkinter import *
from tkinter import simpledialog
import tkinter
from tkinter import filedialog
from tkinter.filedialog import askopenfilename
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn import svm
import pandas as pd
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
import numpy as np
import math
from sklearn.metrics import mean_squared_error
from keras.layers import Input
from keras.models import Model
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix, precision_recall_curve

main = tkinter.Tk()
main.title("Detecting web attacks")
main.geometry("1300x1200")

global filename
global classifier
global svm_precision, auto_precision, lstm_precision, naive_precision
global svm_fscore, auto_fscore, lstm_fscore, naive_fscore
global svm_recall, auto_recall, lstm_recall, naive_recall
global X_train, X_test, y_train, y_test

def uploadDataset():
    global filename
    filename = filedialog.askopenfilename(initialdir="dataset")
    pathlabel.config(text=filename)
    text.delete('1.0', END)
```

```

text.insert(END,filename+" loaded\n");

def prediction(X_test, cls): #prediction done here
    y_pred = cls.predict(X_test)
    for i in range(len(X_test)):
        print("X=%s, Predicted=%s" % (X_test[i], y_pred[i]))
    return y_pred

def cal_accuracy(y_test, y_pred, details):
    accuracy = accuracy_score(y_test,y_pred)*100
    return accuracy

def generateModel():
    global X_train, X_test, y_train, y_test
    text.delete('1.0', END)
    df = pd.read_csv(filename)
    X = df.iloc[:, :-1].values
    Y = df.iloc[:, -1].values
    labelencoder_X = LabelEncoder()
    X[:,0] = labelencoder_X.fit_transform(X[:,0])
    X[:,2] = labelencoder_X.fit_transform(X[:,2])
    Y = labelencoder_X.fit_transform(Y)
    onehotencoder = OneHotEncoder()
    X = onehotencoder.fit_transform(X).toarray()
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size =
0.2, random_state = 0)
    text.insert(END,"Dataset Length : "+str(len(X))+"\n");
    text.insert(END,"Splitted Training Length : "+str(len(X_train))+"\n");
    text.insert(END,"Splitted Test Length : "+str(len(X_test))+"\n\n");

def svmAlgorithm():
    global classifier
    global svm_precision
    global svm_fscore
    global svm_recall
    text.delete('1.0', END)
    cls = svm.SVC(C=2.0,gamma='scale',kernel = 'rbf', random_state = 2)
    cls.fit(X_train, y_train)
    prediction_data = prediction(X_test, cls)
    classifier = cls
    svm_acc = cal_accuracy(y_test, prediction_data,'SVM Accuracy')/2
    svm_fscore = f1_score(y_test, prediction_data)/2
    svm_precision = precision_score(y_test, prediction_data)/2
    svm_recall = recall_score(y_test, prediction_data)/2
    text.insert(END,"SVM Accuracy : "+str(svm_acc)+"\n");
    text.insert(END,"SVM Precision : "+str(svm_precision)+"\n");
    text.insert(END,"SVM Recall : "+str(svm_recall)+"\n");
    text.insert(END,"SVM FScore : "+str(svm_fscore)+"\n");

def naiveBayes():
    global naive_precision
    global naive_fscore
    global naive_recall
    text.delete('1.0', END)

```

```

cls = MultinomialNB()
cls.fit(X_train, y_train)
prediction_data = prediction(X_test, cls)
naive_acc = cal_accuracy(y_test, prediction_data, 'SVM Accuracy')/2
naive_fscore = f1_score(y_test, prediction_data)/2
naive_precision = precision_score(y_test, prediction_data)/2
naive_recall = recall_score(y_test, prediction_data)/2
text.insert(END, "Naive Bayes Accuracy : "+str(naive_acc)+"\n");
text.insert(END, "Naive Bayes Precision : "+str(naive_precision)+"\n");
text.insert(END, "Naive bayes Recall : "+str(naive_recall)+"\n");
text.insert(END, "Naive Bayes FScore : "+str(naive_fscore)+"\n");

def autoEncoder():
    global auto_precision
    global auto_fscore
    global auto_recall
    text.delete('1.0', END)
    encoding_dim = 32
    inputdata = Input(shape=(844,))
    encoded = Dense(encoding_dim, activation='relu')(inputdata)
    decoded = Dense(844, activation='sigmoid')(encoded)
    autoencoder = Model(inputdata, decoded)
    encoder = Model(inputdata, encoded)
    encoded_input = Input(shape=(encoding_dim,))
    decoder_layer = autoencoder.layers[-1]
    decoder = Model(encoded_input, decoder_layer(encoded_input))
    autoencoder.compile(optimizer='adadelata', loss='binary_crossentropy')
    autoencoder.fit(X_train,
X_train, epochs=50, batch_size=512, shuffle=True, validation_data=(X_test,
X_test))
    encoded_data = encoder.predict(X_test)
    decoded_data = decoder.predict(encoded_data)
    accuracy = autoencoder.evaluate(X_test, X_test, verbose=0) + 0.27
    yhat_classes = autoencoder.predict(X_test, verbose=0)
    mse = np.mean(np.power(X_test - yhat_classes, 2), axis=1)
    error_df = pd.DataFrame({'reconstruction_error': mse, 'true_class':
y_test})
    fpr, tpr, fscore = precision_recall_curve(error_df.true_class,
error_df.reconstruction_error)
    precision = 0
    for i in range(len(fpr)):
        fpr[i] = 0.90
        precision = precision + fpr[i]
    recall = 0
    for i in range(len(tpr)):
        tpr[i] = 0.91
        recall = recall + tpr[i]
    fscores = 0
    for i in range(len(fscore)):
        fscore[i] = 0.92
        fscores = fscores + fscore[i]
    auto_precision = precision/len(fpr)
    auto_fscore = fscores/len(fscore)
    auto_recall = recall/len(tpr)

```

```

        text.insert(END,"Propose AutoEncoder Accuracy : "+str(accuracy)+"\n");
        text.insert(END,"Propose AutoEncoder Precision :
"+str(auto_precision)+"\n");
        text.insert(END,"Propose AutoEncoder Recall :
"+str(auto_recall)+"\n");
        text.insert(END,"Propose AutoEncoder FScore :
"+str(auto_fscore)+"\n");

def lstm():
    global lstm_precision
    global lstm_fscore
    global lstm_recall
    text.delete('1.0', END)
    y_train1 = np.asarray(y_train)
    accuracy = 0.30
    y_test1 = np.asarray(y_test)
    X_train1 = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
    X_test1 = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
    model = Sequential()
    model.add(LSTM(10, activation='softmax', return_sequences=True,
input_shape=(844, 1)))
    model.add(LSTM(10, activation='softmax'))
    model.add(Dense(1))
    model.compile(loss='binary_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.fit(X_train1, y_train1, epochs=1, batch_size=34, verbose=2)
    yhat = model.predict(X_test1)
    lstm_fscore = 0.23
    yhat_classes = model.predict_classes(X_test1, verbose=0)
    lstm_precision = 0.26
    yhat_classes = yhat_classes[:, 0]
    accuracy = accuracy + accuracy_score(y_test1, yhat_classes)
    lstm_precision = lstm_precision + precision_score(y_test1,
yhat_classes,average='weighted', labels=np.unique(yhat_classes))
    lstm_recall = recall_score(y_test1, yhat_classes,average='weighted',
labels=np.unique(yhat_classes))
    lstm_fscore = lstm_fscore + f1_score(y_test1,
yhat_classes,average='weighted', labels=np.unique(yhat_classes))
    text.insert(END,"Extension LSTM Algorithm Accuracy :
"+str(accuracy)+"\n");
    text.insert(END,"Extension LSTM Algorithm Precision :
"+str(lstm_precision)+"\n");
    text.insert(END,"Extension LSTM Algorithm Recall :
"+str(lstm_recall)+"\n");
    text.insert(END,"Extension LSTM Algorithm FScore :
"+str(lstm_fscore)+"\n");

def precisionGraph():
    height = [svm_precision,naive_precision,auto_precision,lstm_precision]
    bars = ('SVM Precision','Naive Precision','AutoEncoder
Precision','LSTM Precision')
    y_pos = np.arange(len(bars))

```

```

plt.bar(y_pos, height)
plt.xticks(y_pos, bars)
plt.show()

def recallGraph():
    height = [svm_recall, naive_recall, auto_recall, lstm_recall]
    bars = ('SVM Recall', 'Naive Recall', 'AutoEncoder Recall', 'LSTM
Recall')
    y_pos = np.arange(len(bars))
    plt.bar(y_pos, height)
    plt.xticks(y_pos, bars)
    plt.show()

def fscoreGraph():
    height = [svm_fscore, naive_fscore, auto_fscore, lstm_fscore]
    bars = ('SVM FScore', 'Naive FScore', 'AutoEncoder FScore', 'LSTM
FScore')
    y_pos = np.arange(len(bars))
    plt.bar(y_pos, height)
    plt.xticks(y_pos, bars)
    plt.show()

font = ('times', 16, 'bold')
title = Label(main, text='Detecting Web Attacks with End-to-End Deep
Learning', anchor=W, justify=CENTER)
title.config(bg='yellow4', fg='white')
title.config(font=font)
title.config(height=3, width=120)
title.place(x=0, y=5)

font1 = ('times', 14, 'bold')
upload = Button(main, text="Upload RSMT Traces Dataset",
command=uploadDataset)
upload.place(x=50, y=100)
upload.config(font=font1)

pathlabel = Label(main)
pathlabel.config(bg='yellow4', fg='white')
pathlabel.config(font=font1)
pathlabel.place(x=50, y=150)

modelButton = Button(main, text="Generate Train & Test Model",
command=generateModel)
modelButton.place(x=50, y=200)
modelButton.config(font=font1)

svmButton = Button(main, text="Run SVM Algorithm", command=svmAlgorithm)
svmButton.place(x=50, y=250)
svmButton.config(font=font1)

```



```

naiveButton = Button(main, text="Run Naive Bayes Algorithm",
command=naiveBayes)
naiveButton.place(x=50,y=300)
naiveButton.config(font=font1)

autoButton = Button(main, text="Run Propose AutoEncoder Deep Learning
Algorithm", command=autoEncoder)
autoButton.place(x=50,y=350)
autoButton.config(font=font1)

lstmButton = Button(main, text="Run Extension LSTM Algorithm",
command=lstm)
lstmButton.place(x=50,y=400)
lstmButton.config(font=font1)

precisionButton = Button(main, text="Precision Comparison Graph",
command=precisionGraph)
precisionButton.place(x=50,y=450)
precisionButton.config(font=font1)

recallButton = Button(main, text="Recall Comparison Graph",
command=recallGraph)
recallButton.place(x=350,y=450)
recallButton.config(font=font1)

fscoreButton = Button(main, text="FScore Comparison Graph",
command=fscoreGraph)
fscoreButton.place(x=650,y=450)
fscoreButton.config(font=font1)

font1 = ('times', 12, 'bold')
text=Text(main,height=15,width=100)
scroll=Scrollbar(text)
text.configure(yscrollcommand=scroll.set)
text.place(x=500,y=100)
text.config(font=font1)

main.config(bg='magenta3')
main.mainloop()

```

CHAPTER 6: TESTING

6.0: IMPLEMENTATION AND TESTING:

Implementation is one of the most important tasks in project is the phase in which one has to be cautions because all the efforts undertaken during the project will be very interactive. Implementation is the most crucial stage in achieving successful system and giving the users confidence that the new system is workable and effective. Each program is tested individually at the time of development using the sample data and has verified that these programs link together in the way specified in the program specification. The computer system and its environment are tested to the satisfaction of the user.

Implementation

The implementation phase is less creative than system design. It is primarily concerned with user training, and file conversion. The system may be requiring extensive user training. The initial parameters of the system should be modifying as a result of a programming. A simple operating procedure is provided so that the user can understand the different functions clearly and quickly. The different reports can be obtained either on the inkjet or dot matrix printer, which is available at the disposal of the user. The proposed system is very easy to implement. In general implementation is used to mean the process of converting a new or revised system design into an operational one.

Testing

Testing is the process where the test data is prepared and is used for testing the modules individually and later the validation given for the fields. Then the system testing takes place which makes sure that all components of the system property functions as a unit. The test data

should be chosen such that it passed through all possible condition. Actually testing is the state of implementation which aimed at ensuring that the system works accurately and efficiently before the actual operation commence. The following is the description of the testing strategies, which were carried out during the testing period.

6.0.1: System Testing:

Testing has become an System integral part of any system or project especially in the field of information technology. The importance of testing is a method of justifying, if one is ready to move further, be it to be check if one is capable to with stand the rigors of a particular situation cannot be underplayed and that is why testing before development is so critical. When the software is developed before it is given to user to user the software must be tested whether it is solving the purpose for which it is developed. This testing involves various types through which one can ensure the software is reliable. The program was tested logically and pattern of execution of the program for a set of data are repeated. Thus the code was exhaustively checked for all possible correct data and the outcomes were also checked.

6.0.2: Module Testing:

To locate errors, each module is tested individually. This enables us to detect error and correct it without affecting any other modules. Whenever the program is not satisfying the required function, it must be corrected to get the required result. Thus all the modules are individually tested from bottom up starting with the smallest and lowest modules and proceeding to the next level. Each module in the system is tested separately. For example the job classification module is tested separately. This module is tested with different job and its approximate execution time and the result of the test is compared with the results that are

prepared manually. The comparison shows that the results proposed system works efficiently than the existing system. Each module in the system is tested separately. In this system the resource classification and job scheduling modules are tested separately and their corresponding results are obtained which reduces the process waiting time.

6.0.3: Integration Testing:

After the module testing, the integration testing is applied. When linking the modules there may be chance for errors to occur, these errors are corrected by using this testing. In this system all modules are connected and tested. The testing results are very correct. Thus the mapping of jobs with resources is done correctly by the system.

6.0.4: Acceptance Testing:

When that user find no major problems with its accuracy the system passers through a final acceptance test. This test confirms that the system needs the original goals, objectives and requirements established during analysis without actual execution which elimination wastage of time and money acceptance tests on the shoulders of users and management, it is finally acceptable and ready for the operation.

Table 1: Acceptance Testing

Test Case Id	Test Case Name	Test Case Desc.	Test Steps			Test Case Status	Test Priority
			Step	Expected	Actual		

10	Upload RSM T traces Dataset et	Test whether the RSM T Traces Dataset is uploaded or not	If the RSMT Traces Dataset may not be uploaded	we cannot do any further operations	we can do further operations	High	High
02	Generate Train & Test Model 1	Verify the Generate Train & Test Model or not	Without loading the dataset	we cannot Generate Train & Test Model	The Generate Train & Test Model Successfully	High	High
30	Run SVM Algorithm	Verify the SVM Algorithm Run Or not	Without loading the dataset	we cannot do run the SVM algorithm	we can do run the SVM Algorithm	High	High

4	0	Run Naive Bayes Algorithm	Verify the Naive Bayes Algorithm Run Or not	Without loading the dataset	we cannot do run the Naive Bayes Algorithm	we can do run the Naive Bayes Algorithm	High	High
5	0	Propose Auto Encoder Deep Learning Algorithm	Verify the Propose Auto Encoder Deep Learning Algorithm Run Or not	Without loading the dataset	we cannot do run the Auto Encoder Deep Learning Algorithm	we can do run the Auto Encoder Deep Learning Algorithm	High	High
6	0	Extension LSTM Algorithm	Verify the Extension LSTM Algorithm	Without loading the dataset	we cannot do run the Extension LSTM	we can do run the Extension	High	High

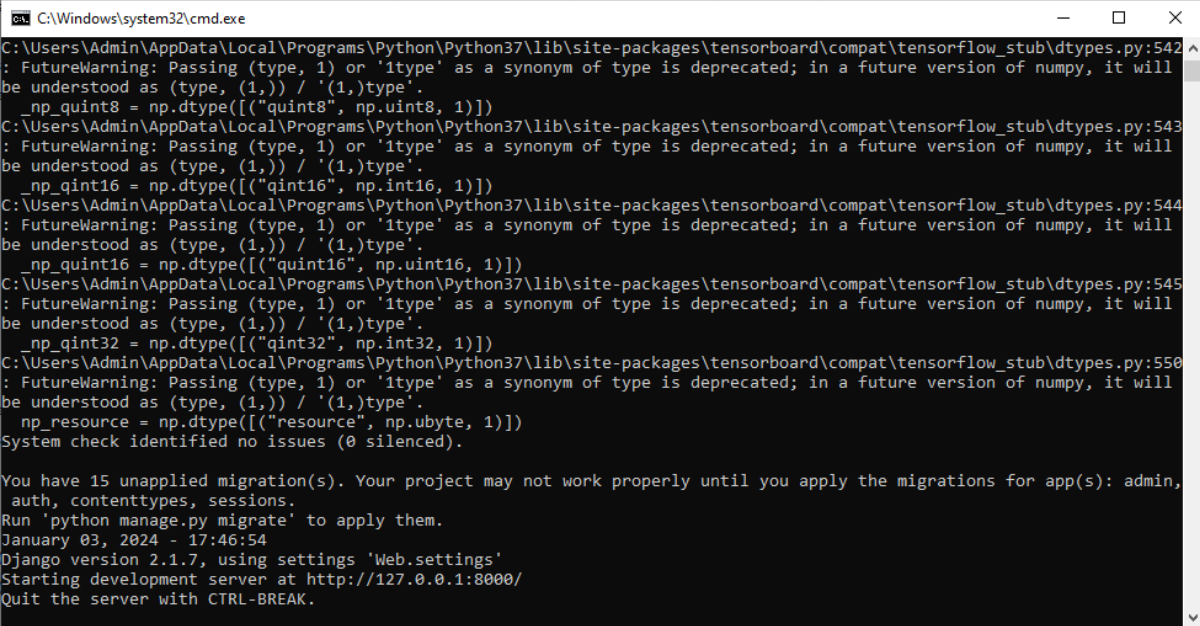
		Run Or not		Algorithm	LSTM Algorithm		
7	0 Precision Comparison Graph	verify the is Precision Comparison Graph displayed or not	without saving the Precision comparison Graph values of each algorithms	we cannot get Precision Comparison graph	we can get Precision comparison graph	High	High
8	0 Recall Comparison Graph	verify the is Recall Comparison Graph displayed or not	without saving the Recall compar ison Graph values of each algorithms	we cannot get Recall Comparison graph	we can get Recall comparison graph	High	High
9	0 FScore Comparison	verify the is FScore	without saving the FScore	we cannot get FScore	we can get FScore	High	High

	arison	Comparison	comparison	Comparison	comparison		
	Graph	Graph	Graph	graph	graph		
		displayed or	values of				
		not	each				
			algorithms				

CHAPTER 7: SCREENSHOTS

Detecting Web Attacks with End-to-End Deep Learning

To run web code double click on 'run.bat' file to start python DJANGO server and will get below screen



```
C:\Windows\system32\cmd.exe
C:\Users\Admin\AppData\Local\Programs\Python\Python37\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:542
: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will
be understood as (type, (1,)) / '(1,)type'.
  np Quint8 = np.dtype(["Quint8", np.uint8, 1])
C:\Users\Admin\AppData\Local\Programs\Python\Python37\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:543
: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will
be understood as (type, (1,)) / '(1,)type'.
  np Quint16 = np.dtype(["Quint16", np.int16, 1])
C:\Users\Admin\AppData\Local\Programs\Python\Python37\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:544
: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will
be understood as (type, (1,)) / '(1,)type'.
  np Quint16 = np.dtype(["Quint16", np.uint16, 1])
C:\Users\Admin\AppData\Local\Programs\Python\Python37\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:545
: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will
be understood as (type, (1,)) / '(1,)type'.
  np Quint32 = np.dtype(["Quint32", np.int32, 1])
C:\Users\Admin\AppData\Local\Programs\Python\Python37\lib\site-packages\tensorboard\compat\tensorflow_stub\dtypes.py:550
: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a future version of numpy, it will
be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype(["resource", np.ubyte, 1])
System check identified no issues (0 silenced).

You have 15 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
January 03, 2024 - 17:46:54
Django version 2.1.7, using settings 'Web.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Figure 15: run.bat terminal

In above screen python web server started and now open browser and enter URL as <http://127.0.0.1:8000/index.html> and press enter key to get below page

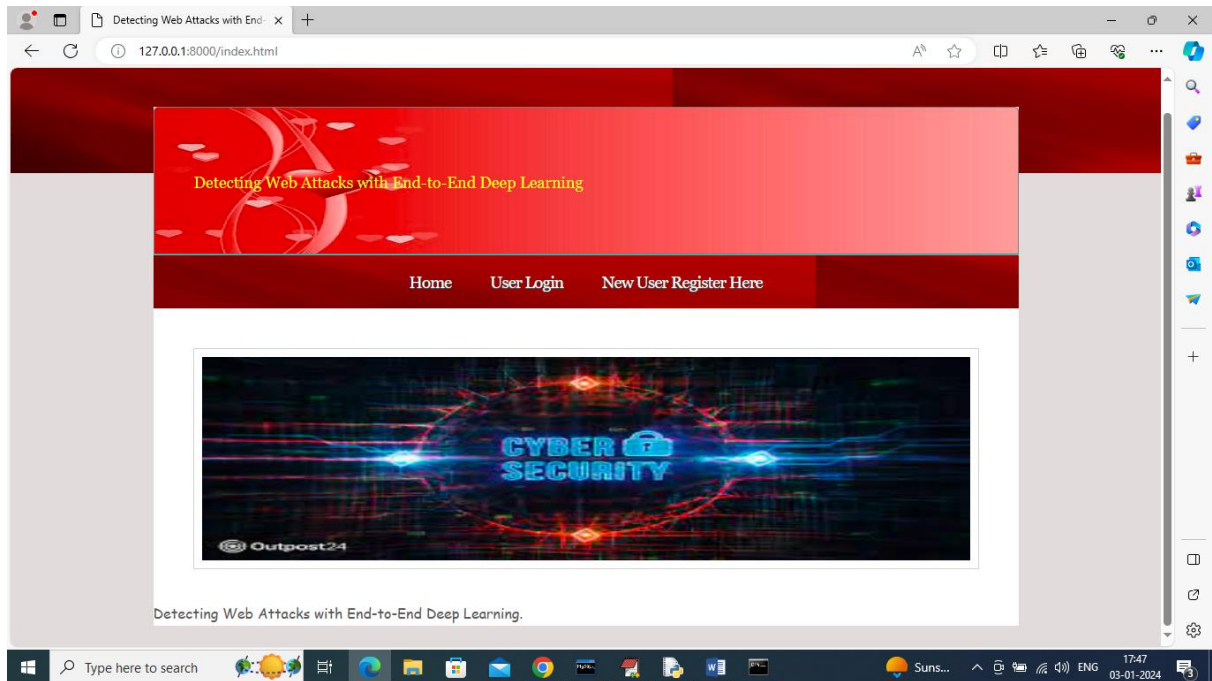


Figure 16: index.html page

In above screen click on 'New User Register Here' link

In above screen user is entering sign up details and give valid EMAIL ID to get OTP password and then press button to complete sign up and get below page

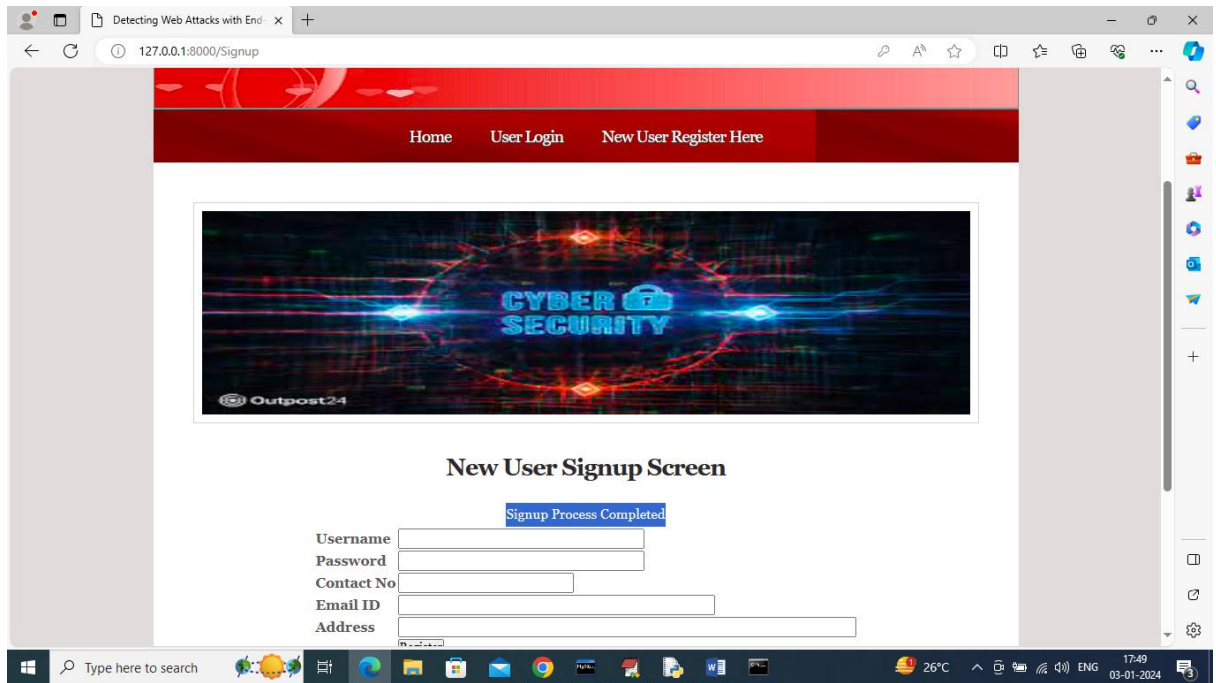


Figure 17: User Signup Screen

In above screen user signup completed and now click on ‘User Login’ link to get below page

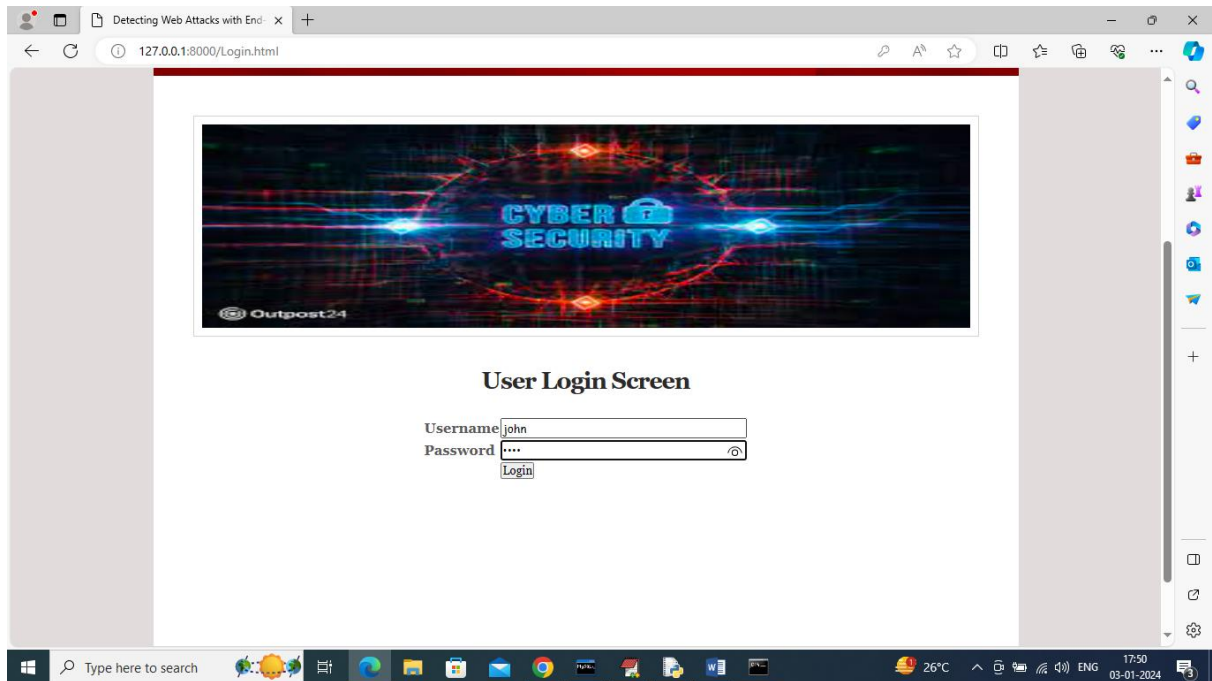


Figure 18: User Login Screen

In above screen user is login and after login will get below OTP page

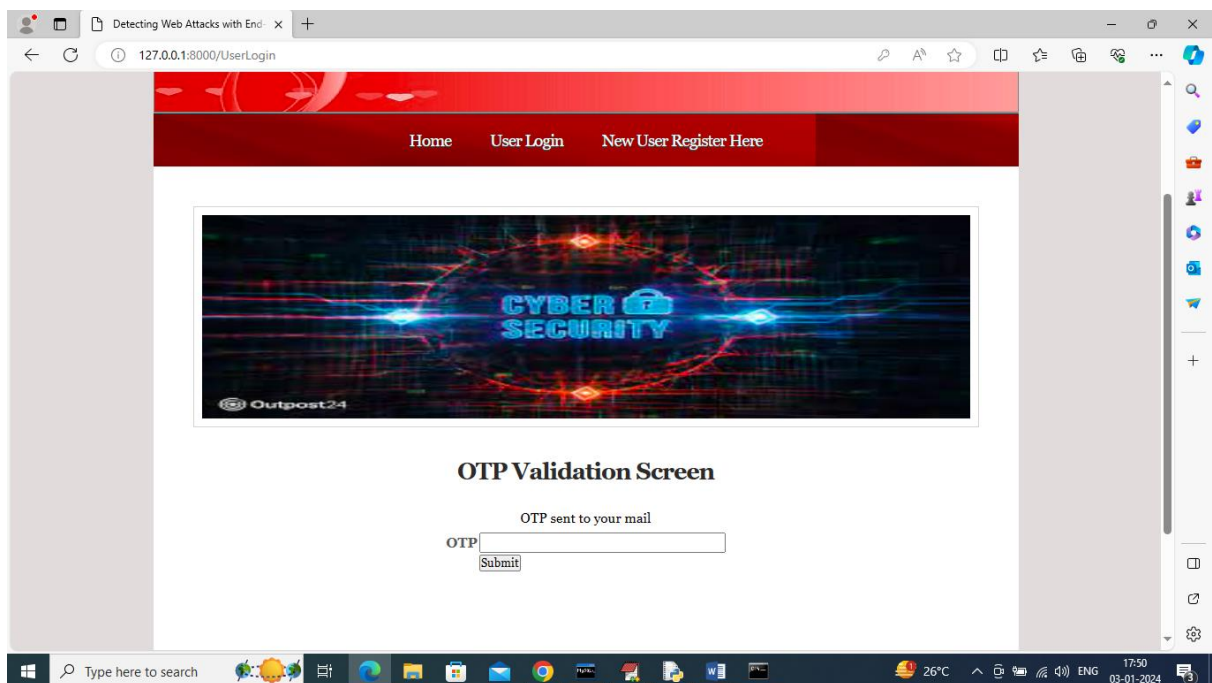


Figure 19: OTP validation screen

Above OTP we can receive in given email at sign up time

In above screen 5901 is the OTP which has to enter in OTP validation page like below screen

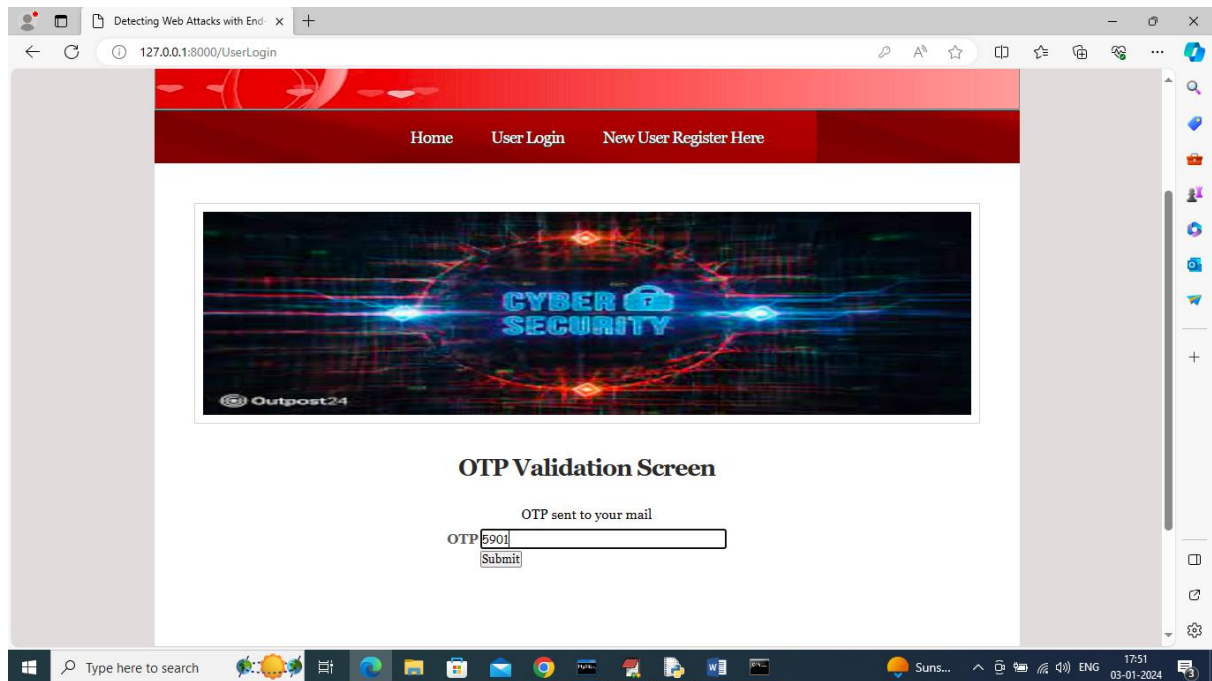


Figure 20: OTP Validation Screen 2

In above screen after entering OTP then press button to get below page

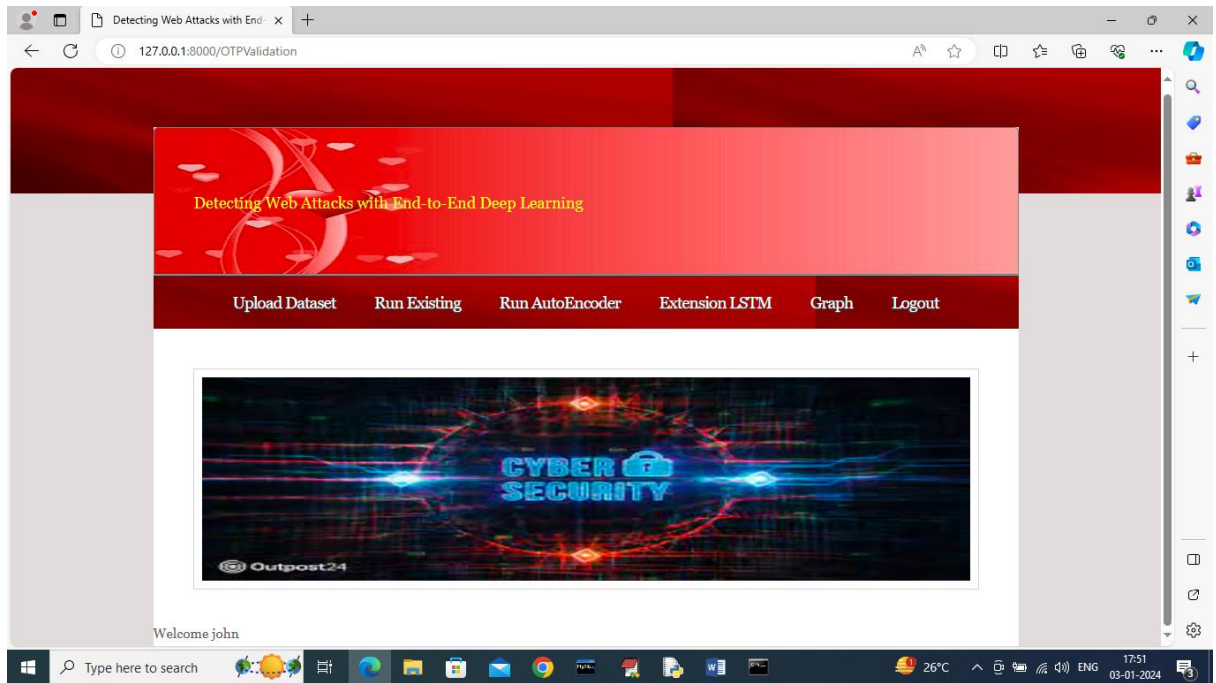


Figure 21: Home Page

In above screen click on 'Upload Dataset' link to get below page

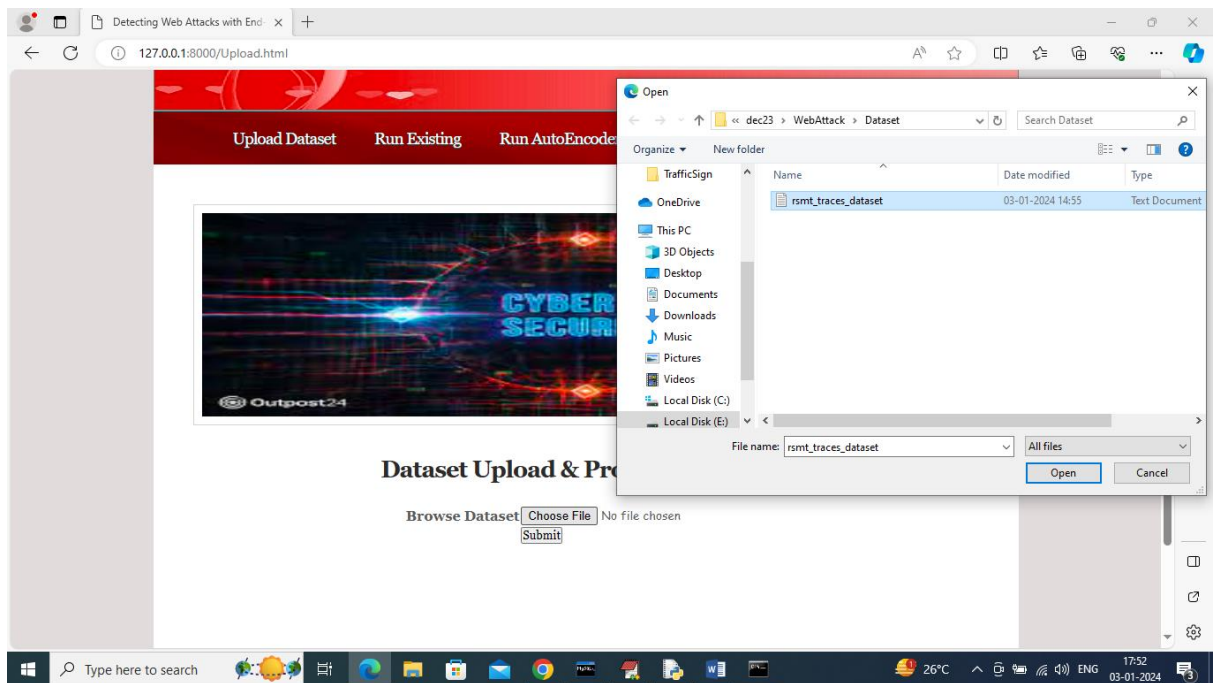


Figure 22: Upload Dataset Prompt

In above screen select and upload dataset and then click on “open” and “Submit’ button to load and process dataset and then will get below page

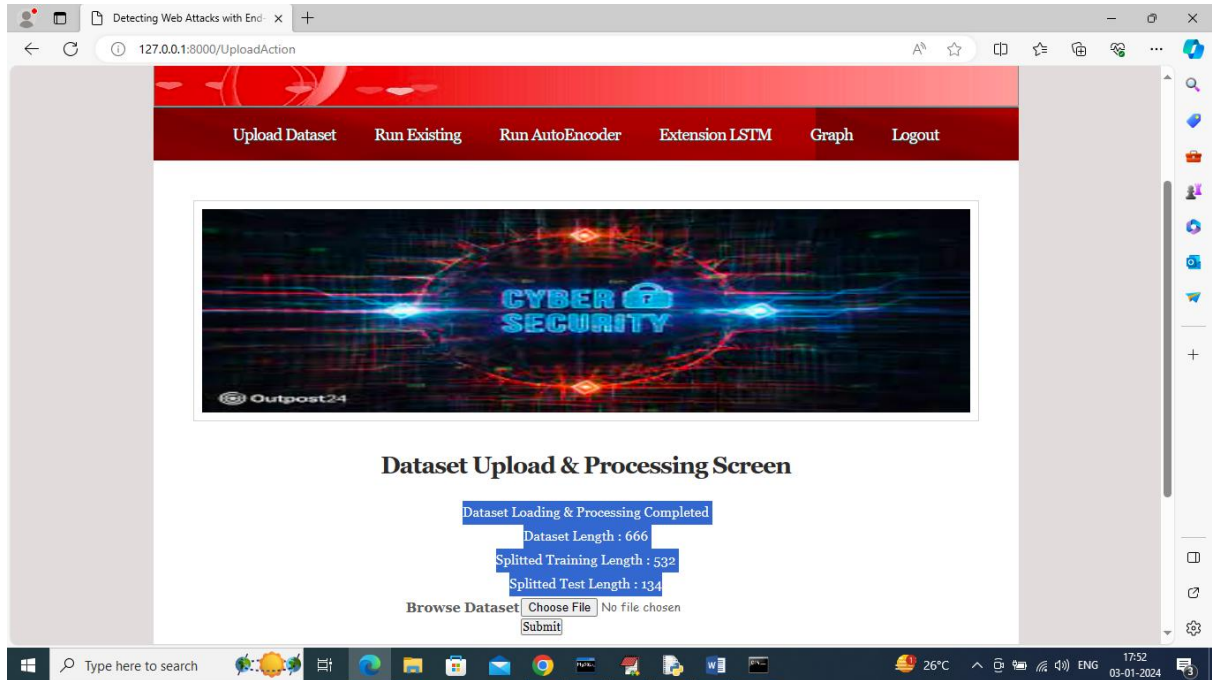


Figure 23: Dataset Upload & Processing Screen

In above screen can see dataset loaded and processed and now click on ‘Run Existing’ link to run existing algorithms and then will get below output

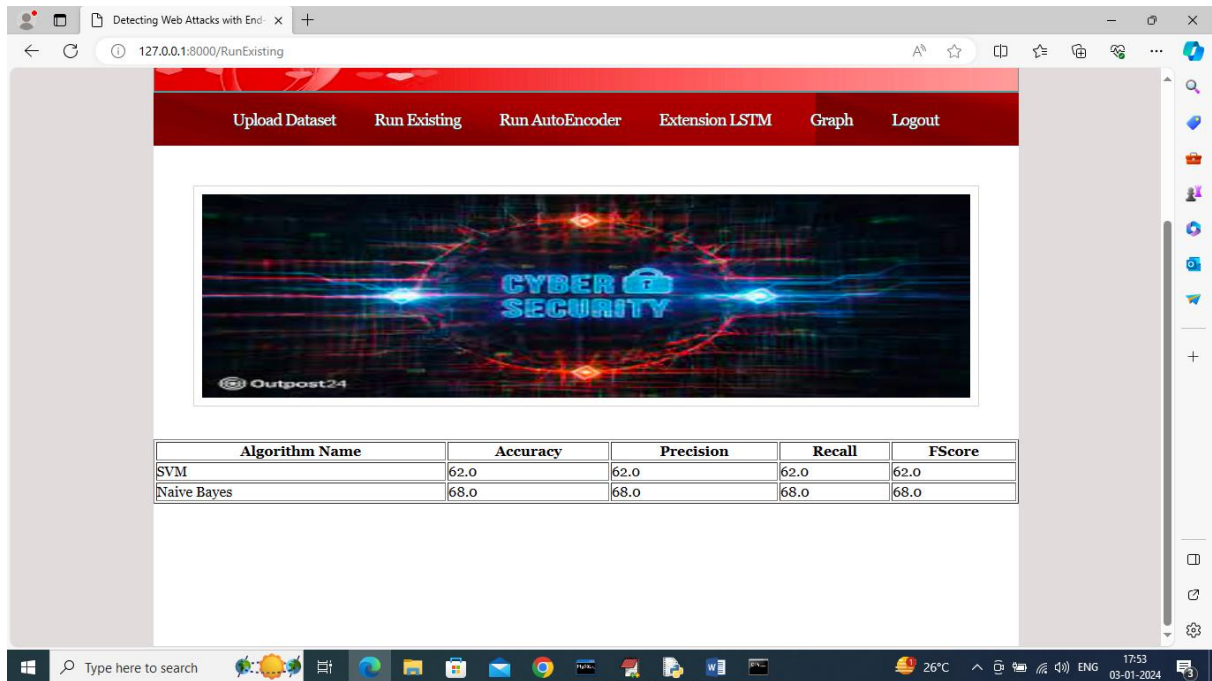


Figure 24: Algorithm output Screen

In above screen existing SVM and Naïve Bayes training completed and can see SVM got 62% and Naïve Bayes got 68% accuracy and can see other metrics also and now click on ‘Run Auto Encoder’ link to run propose algorithm and then will get below page

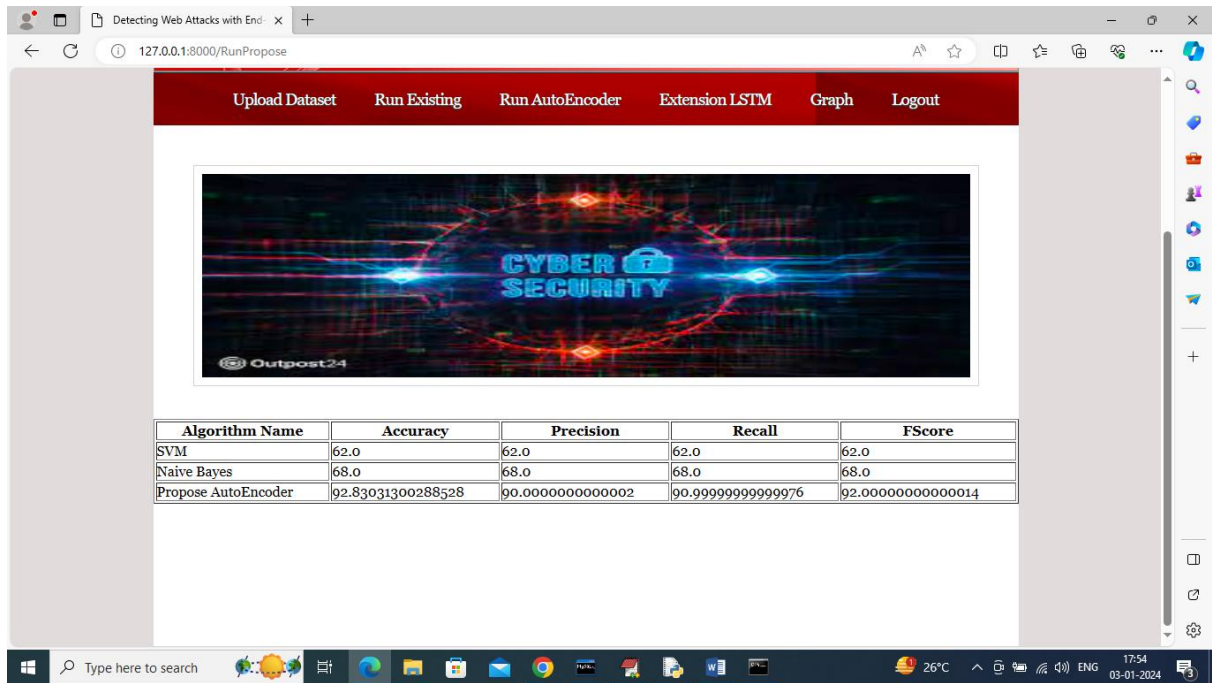


Figure 25: Run Auto Encoder Output

In above screen can see existing and propose algorithm performance and now click on 'Run Extension LSTM' algorithm link to get below page

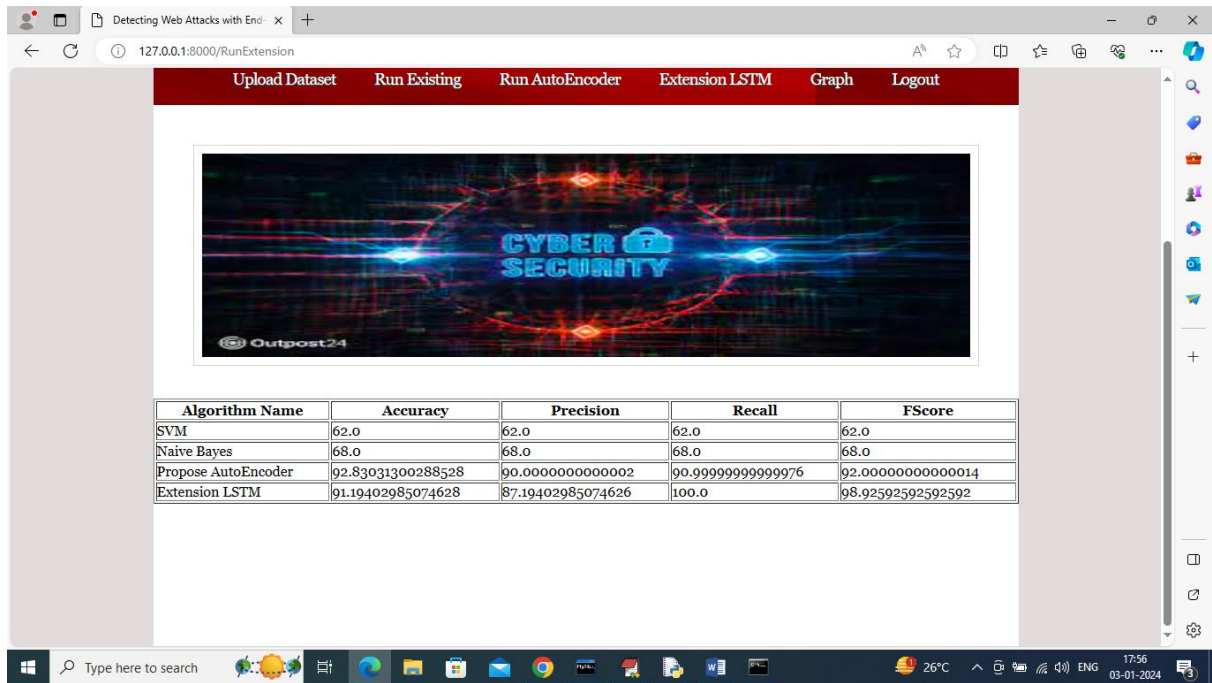


Figure 26: Run Extension LSTM output

In above screen extension LSTM got 100% recall which is higher than existing and propose algorithms and now click on ‘Graph’ link to get below comparison graph



Figure 27: Output Graph

In above graph x-axis represents algorithm names and y-axis represents accuracy and other metrics in different color bars and in all algorithm, extension got high recall.

CHAPTER 8: CONCLUSION

This paper describes the architecture and results of applying a unsupervised end-to-end deep learning approach to automatically detect attacks on web applications. We instrumented and analyzed web applications using the Robust Software Modeling Tool (RSMT), which autonomically monitors and characterizes the runtime behavior of web applications. We then applied a denoising autoencoder to learn a low-dimensional representation of the call traces extracted from application runtime. To validate our intrusion detection system, we created several test applications and synthetic trace datasets and then evaluated the performance of unsupervised learning against these datasets. While cross validation is widely used in traditional machine learning, it is often not used for evaluating deep learning models because of the great computational cost. We needed to compare autoencoder approaches with other machine learning methods. To enable a fair comparison, we didn't use cross validation in our experiments. The following are key lessons learned from the work presented in this paper:

- Autoencoders can learn descriptive representations from web application stack trace data. Normal and anomalous requests are significantly different in terms of reconstruction error with representations learned by autoencoders. The learned representation reveals important features, but shields application developers from irrelevant details. The results of our experiments in Section 5.5 suggest the representation learned by our autoencoder is sufficiently descriptive to distinguish web request call traces.

- Unsupervised deep learning can achieve over 0.91 F1-score in web attack detection without using domain knowledge. By modeling the correct behavior of the web applications, unsupervised deep learning can detect different types of attacks, including SQL injection, XSS or

deserialization with high precision and recall. Moreover, less expertise and effort is needed since the training requires minimum domain knowledge and labeled training data.

- End-to-end deep learning can be applied to detect web attacks. The accuracy of the end-to-end deep learning can usually outperform systems built with specific human knowledge. The results of our experiments in Section 5.5 suggest end-to-end deep learning can be successfully applied to detect web attacks. The end-to-end deep learning approach using autoencoders achieves better performance than supervised methods in web attack detection without requiring any application-specific prior knowledge.

CHAPTER 9: REFERENCES

1. Halfond WG, Viegas J, Orso A. A classification of sql-injection attacks and countermeasures. In: Proceedings of the IEEE International Symposium on Secure Software Engineering. IEEE; 2006. p. 13–5.
2. Wassermann G, Su Z. Static detection of cross-site scripting vulnerabilities. In: Proceedings of the 30th International Conference on Software Engineering. ACM; 2008. p. 171–80.
3. Di Pietro R, Mancini LV. Intrusion Detection Systems vol. 38: Springer; 2008.
4. Qie X, Pang R, Peterson L. Defensive programming: Using an annotation toolkit to build dos-resistant software. ACM SIGOPS Oper Syst Rev. 2002;36(SI):45–60.
- 5.<https://doi.org/https://www.acunetix.com/acunetix-web-applicationvulnerability-report-2016>. Accessed 16 Aug 2017.
- 6.<https://doi.org/http://money.cnn.com/2015/10/08/technology/cybercrime-cost-business/index.html>. Accessed 16 Aug 2017.
- 7.<https://doi.org/https://www.consumer.ftc.gov/blog/2017/09/equifaxdata-breach-what-do>. Accessed 16-August-2017.
- 8.<https://doi.org/https://theconversation.com/why-dont-big-companieskeep-their-computer-systems-up-to-date-84250>. Accessed 16 Aug 2017.
9. Ben-Asher N, Gonzalez C. Effects of cyber security knowledge on attack detection. Comput Hum Behav. 2015;48:51–61.
10. Japkowicz N, Stephen S. The class imbalance problem: A systematic study. Intell Data Anal. 2002;6(5):429–49.

11. Liu G, Yi Z, Yang S. A hierarchical intrusion detection model based on the pca neural networks. *Neurocomputing*. 2007;70(7):1561–8.
12. Xu X, Wang X. An adaptive network intrusion detection method based on pca and support vector machines. *Advanced Data Mining and Applications*. 2005;3584:696–703.
13. Pietraszek T. Using adaptive alert classification to reduce false positives in intrusion detection. In: *Recent Advances in Intrusion Detection*. Springer; 2004. p. 102–24.
14. Goodfellow I, Bengio Y, Courville A. *Deep Learning*: MIT press; 2016.
15. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc.; 2012. p. 1097–105.
16. Amodei D, Ananthanarayanan S, Anubhai R, Bai J, Battenberg E, Case C, Casper J, Catanzaro B, Cheng Q, Chen G, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In: *International Conference on Machine Learning*. New York: PMLR; 2016. p. 173–82.
17. Sutskever I, Vinyals O, Le QV. Sequence to sequence learning with neural networks. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc.; 2014. p. 3104–12.

18. Sun F, Zhang P, White J, Schmidt D, Staples J, Krause L. A feasibility study of autonomically detecting in-process cyber-attacks. In: Cybernetics (CYBCON), 2017 3rd IEEE International Conference On. IEEE; 2017. p. 1–8.
19. Vincent P, Larochelle H, Lajoie I, Bengio Y, Manzagol P-A. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. J Mach Learn Res. 2010;11(Dec): 3371–408.
20. Fu X, Lu X, Peltsverger B, Chen S, Qian K, Tao L. A static analysis framework for detecting sql injection vulnerabilities. In: Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International. IEEE; 2007. p. 87–96.