A Secured Cloud System based on Log Analysis

Sindhusha Doddapaneni
San Jose State University

A Secured Cloud System based on Log Analysis

A Writing Project Report

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

By

Sindhusha Doddapaneni

DECEMBER 2015

The Designated Project Committee Approves the Project Titled

A Secured Cloud System based on Log Analysis

By

Sindhusha Doddapaneni

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

DECEMBER 2015

ABSTRACT

Now-a-days, enterprises' acceptance over the Cloud is increasing but businesses are now finding issues related to security. Everyday, users store a large amount of data in the Cloud and user input may be malicious. Therefore, security has become the critical feature in the applications stored in the Cloud. Though there are many existing systems which provide us different encryption algorithms and security methods, there is still a possibility of attacks to applications and increasing data modifications. The idea behind this project is to find attacks and protect the applications stored in the Cloud using log analysis. The proposed solution detects the SQL injection attack, which is supposed to be the most critical security risk of vulnerable applications. The goal of this research is to detect the SQL injection attacks for an application stored in the Cloud by analyzing the logs. To achieve this, the proposed system automates the intrusion detection process for an application by performing log analysis. Log Analysis is performed by combining the implementation of two different methodologies called learn and detect methodology and pattern recognition system. The accuracy of SQL injections detected on log data is dependent on the order in which these two methodologies are applied. The outcome after applying these two methodologies results in information which helps a security analyst to understand and know the root cause of every attack that is detected on an application.

ACKNOWLEDGEMENTS

I would like to express sincere gratitude to my advisor Dr. Melody Moh for her guidance and support throughout the research process. I am very thankful to her for giving me an opportunity to work on the project in a field of my personal interest.

A very special thanks to my committee member Dr. Teng Moh for his suggestions and valuable time to monitor the progress of the project very closely. I would also like to thank my other committee member Mr. John Cribbs for his time and support.

Furthermore, thanks to my family and friends for the encouragement given in every step of the way throughout my master's project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS

IaaS: Infrastructure as a Service

PaaS: Platform as a Service

SaaS: Software as a Service

API: Application Programming Interface

SQL: Structured Query Language

IPLoM: Iterative Partitioning Log Mining

Amazon EC2: Amazon Elastic Cloud Compute

Amazon S3: Amazon Simple Storage Service

Amazon RDS: Amazon Relational Database Service

Amazon SQS: Amazon Simple Queue Service

Weka: Waikato Environment for Knowledge Analysis

ARFF: Attribute Relation File Format

CHAPTER 1

INTRODUCTION

1.1 Introduction to Cloud Security

In recent years, the Cloud has become a buzzword in enterprises because of availability, scalability, and economic business objectives. A primary issue with the Cloud is security. Cloud service providers don't uniformly guarantee security. There are security concerns which are divided into two categories, namely security issues faced by Cloud service providers (provides Cloud services such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS)) and security issues faced by customers (individuals or any organizations may deploy their applications and store data on the Cloud [10]).

These days, applications hosted on a Cloud are prone to attacks and some of the security issues are due to unauthorized access, server hacking, etc.

The main security issues are [10]:

Data Breaches: This means some other person has gained access to the data of the authorized user. Unauthorized logins are possible in the applications hosted on the Cloud.

Data Loss: Third-party servers store data in the Cloud and give access to the customers. Therefore, the data can be lost due to damage or server hacking.

Account Hijacking: The Cloud provides online accounts to users where data can be accessed. In such cases, some of the accounts are compromised by any hacker and the confidential data will be at risk.

Insecure API's: Application Programming Interface (API) is used to call and manage Cloud data. Malicious data transfer is possible in this type of issue.

Denial of Service: In this attack, the authorized user is blocked from getting services like data access, etc.

1.2 Problem Description

Enterprises now-a-days generate a huge amount of data each day. The automatic increase in data is possible when enterprises start recruiting people every day when new software is built and during software deployment on the Cloud. Such kind of data contains confidential and interesting information in high volumes, which must be secured. Day-by-day major security attacks make news every day, so we need to take steps to address the main cause of the attacks. In existing systems, there are different encryption algorithms and security methods to protect data retrieved from applications, but there is still a chance of attacks to applications and incremental data modifications. Since there is an easy access to applications stored in the Cloud, they are easily prone to vulnerabilities. The attackers can easily perform illegal activities in a web application, which leads to data loss and data modifications.

The most commonly seen attack that performs illegal activities on Cloud applications is an SQL injection attack.

SQL injection attack is seen in most web applications where the attacker fills in the Structured Query language code as an input in order to gain access or make changes to a database [12]. Based on the main agenda of the attack, attackers can inject a variety of methods of SQL injections to web application input forms. An example of SQL injection attack is unauthorized access to applications. Unauthorized access can be achieved by using the tautology class in SQL injection. The SQL statement is modified in a way that whatever password is given, login is possible for the attackers. Here is the SQL query to retrieve data from a database:

select * from table where username='user' and password='pass'

If the input of the user is 'OR 1=1'—'malicious statement, then it is injected into the above non-malicious SQL statement. After the injection, where the clause statement each time is true because of the tautology 1=1 is used, the attacker can login to the application.

To protect data from such attacks, there are many log analysis mechanisms which are used to find the main cause of an attack, but no one used live detection solution for protecting an application from such attacks. Manual log analysis is not possible because the amount of data generated is huge and this also involves more cost and time. A secured system is required to perform real-time log analysis, which helps security analyst for further investigation.

This paper is explained as follows: Chapter 2 gives a review of related research work; Chapter 3 gives the system design; Chapter 4 explains the implementation details and also the discussion about the comparisons of Kibana and BayesNet classifier methods; Chapter 5 explains the relevant work done for experiment and results of the experiments done using the proposed system; Chapter 6 compares the two detection models and also gives reasons for results obtained using the combination of two detection models; Chapter 7 gives the conclusion and the future work related to the project.

CHAPTER 2
BACKGROUND

2.1 Research Work

This research work is on issues related to the security of data present in the Cloud. As the sensitive information and data are shared with the third-party providers, Cloud users try to avoid an untrusted Cloud provider. Protecting sensitive and confidential information from attackers is of major importance. To supplement this, my research work is focused on how to detect security attacks for the applications stored in a Cloud system. One of the attacks that is difficult to detect and easy to inject is an SQL injection attack. Currently, there are many systems built for detecting and preventing SQL injection attacks. The most common approaches available for detecting SQL injection attacks are parsing techniques, machine learning techniques, and pattern matching techniques.

Everyone spends most of their time on understanding and parsing the logs. This research paper provides a concept of computing the similarity score between the logged events and a collection of important events [9]. This paper explains how to calculate the similarity values in order to reduce the count of the log. It will also describe the stated log modeling and preprocessing, the evaluation of event similarity approaches and log filtering, and various caching algorithms. As a future work, we are considering implementing feature reduction techniques to improve performance and enhance the results, as well as the improvement of clustering or pattern matching techniques, which take inter-event relationships into consideration.

To understand the log data formats I have gone through, this research paper outlines different types of data, different approches to converting log data into semantic format, log formats used for different operating systems, and log conversion tools [17]. Authors referenced in this paper suggested some semantic technologies which perform meaningful log analysis.

The existing systems should be able to handle large volumes of data for intrusion detection. In this research paper, the proposed design handles large volumes of data and also processes high-volume data using Hadoop, Map-reduce and Cloud computing concepts [11]. Once data is collected, it’s then analysed to determine if any malicious attacks are present. The main focus of this paper is to improve intrusion detection system scalability and throughput in order to perform log analysis.

Additionally, if there is a high volume of log data, then the system demands new frameworks and techniques of computing. A research paper that captured my attention proposes a lightweight distributed log analysis framework that allows organizations to analyze a high number of logs [15]. This framework makes use of dynamic task scheduling to stream log data. Moreover, the distributed log analysis framework used Amazon web services to develop analysis applications which help in calculating security event occurrences.

Most organizations face challenges in correlating different sources of data and extracting meaningful information from that data. One research paper proposes a system with multi-core SAL architecture and also addresses the challenges by using various K-means-based correlation algorithms, clustering algorithms such as ROCK and QROCK algorithms and integrated into multicore SAL architecture [4].

To understand the content on log data, another research paper proposes an algorithm called IPLoM Iterative Partitioning Log Mining, which performs message-type extraction from log files [13]. Message types are a semantic grouping of system log messages. This paper states that IPLoM is a three-step hierarchical process which partitions logs into groups of event log messages, and discovers the message-type description for each partition. Once message types are determined, these not only provide groupings for categorizing and summarizing log data, but also provide help in visualization. This methodology is useful for log analysis even though the log consists of other fields.

This research paper describes the different web application vulnerabilities which are very easy to take advantage of and difficult to detect in Cloud applications [8]. This paper proposes a system where an algorithm called Boyer Moore String Matching algorithm is used on user inputs, which in turn helps to detect the security attacks and produces a report regarding detection of attacks.

Besides the conference papers, I have also performed some background work to know more about the ways to detect security attacks. This research work explains the steps on how to collect the data, and how to extract meaningful information from the logs using Log parsing concepts. This research provides us a better understanding of log analysis. Furthermore, I also went through some of the websites which describe how to secure the Cloud using data analytics. This provides a better understanding of the disadvantages of traditional analytics and also compares the two different techniques: traditional analytics and big data analytics.

My research work on security attack detection methods for the application stored in the Cloud has given me an ample amount of information to move forward with the research in a similar field. Based on the research, the proposed system detects SQL injection attacks by performing real-time log analysis of streaming data, which is discussed in the next sections.

2.2 Used Tools/Technologies

Apache Log4j

Log4j is a Java framework used for logging [6]. It demonstrates the logging process in terms of log level priorities and offers methods to direct the log information to different places like a database, file, console, UNIX sys log, etc. It supports multiple log levels such as DEBUG, ERROR, INFO, WARN, and FATAL.

Log4j has three main components [6]:

a) **Loggers:** Loggers are responsible for capturing logging information.
b) **Appenders:** Appenders are responsible for forwarding logging information to various places such as a console, file, and database, etc. There are different types of Appenders based on the destination specified, such as Console Appender, File Appender, and Flume Appender. File Appender is the most commonly used appender in web applications to write the log information to a file, and this is achieved by using org.apache.log4j.FileAppender. Logging can be done in multiple files when a file size reaches a particular size limit, then RollingFileAppender is used.
c) **Layouts:** Layouts are responsible for formatting the logging information supported by Log4j. Various layout objects are provided by log4j which can format logging information according to the layout subclass selected. There are different types of subclasses for layouts such as Date Layout, Pattern Layout, HTML layout, Simple Layout, and XML Layout.

Amazon Web Services

This is a set of services offered by Amazon called Web Services. Across the world, these services are based on 11 geographical regions. Amazon Web Services provide IT resources to people when on demand with pay-as-you-go service [20]. Amazon Web Services offer a set of compute, storage, database, application, and deployment services.

Amazon EC2: Within Amazon Web Services, computing services provides the set of instances which can automatically scale up and down to meet the needs of an application. One of the services is Amazon EC2. The Amazon EC2 web service provides users with both growable computing capacity in the Cloud and also gives users different tools to build fault-tolerant applications. The Amazon EC2 web service allows users to create instances where there is a possibility of scalable deployment of applications. With respect to payments for active servers, a user can construct, launch, and stop the instances.

Benefits of Amazon EC2 [20]:

Scalable Computing: The Amazon EC2 web service has a web service API which allows users to increase or decrease the capacity within minutes.

In Complete Control: Users have complete control over instances and can interact with them. Users can stop an instance and can subsequently restart the same instance using a web service API.

Flexible Service: This allows users to select the configuration of memory, choice of operating system, CPU, and storage for instances and software packages.

Interaction with other web services: To have a mix of different services like computing, query processing, and storage in an application, Amazon EC2 works with Amazon S3, Amazon RDS, Amazon Simple DB, Amazon SQS services.

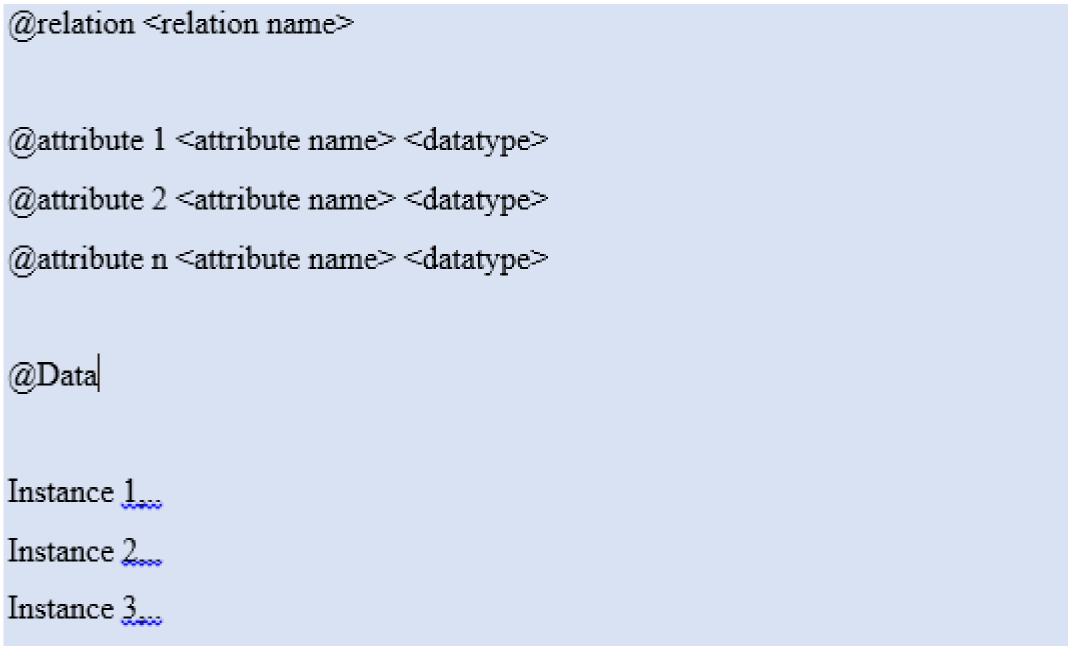
Inexpensive: Amazon allows you to pay a very low rate on usage capacity. Additionally, it has a usage-based payment service where the user pays on demand without any further committed payments.

Secure: Amazon EC2 service users can decide which instances should be on the internet and which instances should remain private.

Weka

Weka is an open-source software implemented in Java programming language to discover interesting information on large datasets [18]. It provides a set of machine learning algorithms to apply on different datasets in order to perform data mining tasks. Furthermore, these different algorithms can be implemented on the datasets directly or by calling our own Java code. Weka only accepts datasets which have ARFF and CSV file formats. Weka provides tools for pre-processing, classifying data, association rules, regression, clustering data, and visualization.

ARFF: Attribute Relation File Format file was developed at the University of Waikato for use with Weka software. ARFF [19] file is a text file which consists of a list of instances sharing a list of attributes. ARFF file format consists of two sections: header section and data section. The header section consists of relation name and a set of attributes with their datatypes. Here, datatype can be of four types: numeric, nominal-specification, string, and date formatted "yyyy-MM-dd'T'HH:mm:ss". The data section of ARFF file format consists of a data declaration line and an instance line. Here, each instance is represented on a single line. The delimiters for the attributes in instances are separated by commas and should be in the order of the attributes declared in the header section.
ARFF file format example:



```
@relation <relation name>

@attribute 1 <attribute name> <datatype>
@attribute 2 <attribute name> <datatype>
@attribute n <attribute name> <datatype>

@Data

Instance 1...
Instance 2...
Instance 3...
```

Supervised Learning

Supervised Learning is a type of machine learning task which takes the known input dataset, aiming to build a predictor model which gives the prediction to the new dataset [3].

Supervised learning is performed in two steps:

Learning: Learn a model using a training dataset.

Testing: Test the model using the new dataset in order to test the accuracy of a model.

Accuracy = Number of correctly classified instances/Total number of test data instances

There are many machine learning algorithms that are available, each with their own strengths and weaknesses. In machine learning, there is no single algorithm that works best for the supervised learning algorithms.

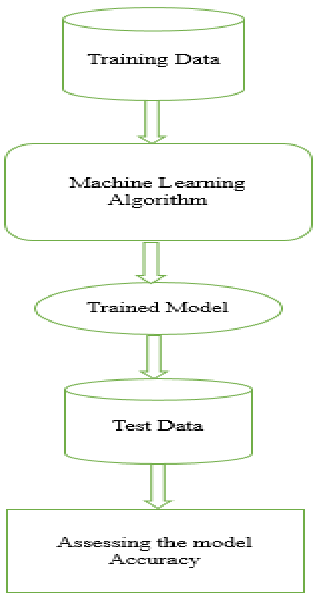


Figure 1: Steps for Supervised Learning Technique

BayesianNet Classifier

The problem arising in using a Naïve-Bayes classifier is its assumption of treating all attributes as strongly independent (i.e. probabilistic independence) of each other [2]. This assumption of being conditionally independent of each other seems unrealistic, as it cannot be applied in situations where correlation exists between these attributes and unwarranted data needs to be ignored to improve performance. Therefore, in order to tackle this problem of efficiently and effectively representing and manipulating the independence assertions, Bayesian Networks are supportive and improves performance.

Bayesian Networks are directed acyclic graphs that represent joint probability distribution over a set of random variables U={x1, x2,…….,xn}, where n>=1 in a problem domain [5]. Each variable lies at every vertex in the graph and the edges form the correlations between these random variables. The conditional independence between these variables is stated in a way that every variable is independent of its non-descendants, considering already the state of their parent variables.

Assumptions:

The buildClassifier checks if the dataset, on which the BayesNet algorithms are implemented in Weka, fulfills the following assumptions [1]:

1. Discrete and Random feature should be present in all the variables. If continuous variables are present in the dataset, then to discretize them; *Discretize filter* is used.

2. None of the instances present in the dataset should have missing values. If missing values are present, *ReplaceMissingValues* filter should be used to fill the missing values.

Logstash

Logstash is a data pipelining tool which connects to a variety of sources with the help of plugins, and it streams data to an analytics system [24]. Logstash receives different types of logs: system logs, web server logs, error logs, and application logs. Today's enterprises have crucial data which is distributed in different formats among different systems. Logstash helps users parse data into one common format before storing into an analytics data store. Additionally, Logstash provides a way to parse custom format logs by providing custom logic to it.

Elasticsearch

Elasticsearch is an open-source search and data analysis software which gives users a deep insight on stream data [24]. The tool provides a scalability feature by allowing users to add new nodes which make the cluster easy to add new hardware. By default, Elasticsearch clusters are resilient; they will detect both new and failed nodes, and automatically reorganize data to ensure the data is safe. Once a cluster is set up, Elasticsearch provides the search and analysis feature by building a distributed environment on the top of Apache Lucene, which is used for full-text searching. Moreover, Elasticsearch is a document-oriented software where all the entities are structured JSON documents and all fields are indexed by default to easily get complex results at high speed. When a JSON document is indexed, Elasticsearch automatically detects the data types and makes the data search easy. Finally, Elasticsearch provides data safety by logging the information whenever there is a document change.

Apache Lucene

Apache Lucene is an open-source Java-based text search engine library [19]. Apache Lucene permits users to write their customized queries through its query API, which helps users to search Geo IP locations, perform multilingual searches, etc. Furthermore, Lucene provides different types of searches, like term and phrase search, and allows users to group keywords for detailed text searching.

Kibana

Kibana is an open-source data visualization interface for a real-time summary and charting of stream data [24]. This tool also provides users with different visualizations which can be combined into custom dashboards. Kibana helps users understand large volumes of datasets by providing different visualization tools like bar charts, pie charts, line diagrams, and geographical maps. Additionally, this increases the power of Elasticsearch to analyze data, to perform mathematical calculations, and to break the data into pieces for further analysis. Visualizations can be done from various sources that pushed data into Elasticsearch from Logstash, Hadoop, Apache flume, and many others. Also, Kibana has sharing options: an export feature to download interesting bits of raw data from custom dashboards, and a link to share data online.

CHAPTER 3

DESIGN

This chapter describes the proposed solution and its architecture. Also, it explains the modules of the system built.

3.1 Proposed Solution

The proposed system performs real-time log analysis for streaming data in order to protect an application from SQL injection attacks. Log analysis is a security mechanism which helps us find out the root cause of the attacks, how an intruder attacked an application, and what steps that intruder performed in order to attack an application. To achieve this, first, a Cloud application was created to get input from users. Second, logs were collected from Cloud application activities using a Java framework called Log4j framework. Finally, log analysis was performed using two different models: learn and detect methodology and Pattern Matching system.

In Learn and detect methodology, BayesNet classifier is used to classify the logs as Malicious or Non-Malicious logs. Based on the classification, the analyst can further investigate the particular user activity. In a Pattern matching system, real-time log analysis is performed for streaming data. Here, a dashboard is created for the security analyst to know the root cause of security attack. The proposed model considers two different methodologies in parallel for log analysis; it also considers the order of two different methodologies applied on logs matters while considering the SQL injection detection accuracy.

3.2 System Architecture

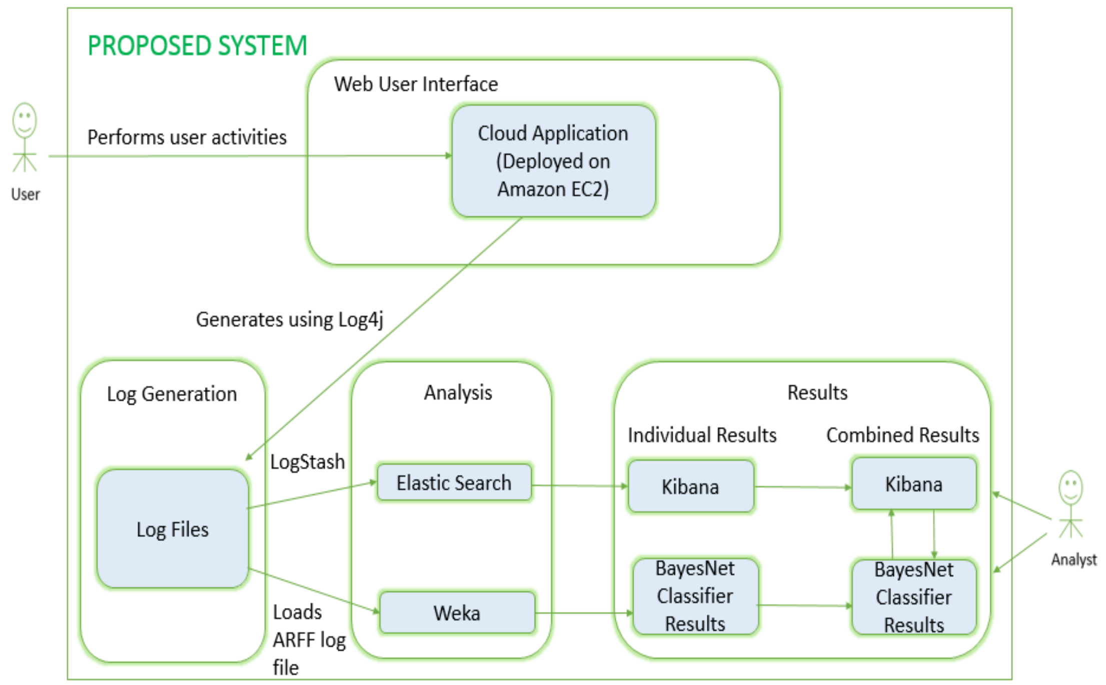


Figure 2: Proposed System Architecture

Below is the description for modules present in the proposed architecture:

Web User Interface module: The web application was created in Java and deployed on Amazon EC2. This is the user interface where users can perform different activities on the web application.

Log Generation module: This is a second module where logs were generated using Log4j framework from the Cloud application and were stored on an Linux EC2 instance.

Analysis module: This is the third module in this project, where the log stash was used to collect logs present in the log file and send it to Elasticsearch tool. The log file was converted to the ARFF file format and sent to Weka tool for data analysis.

Results module: The final module has both individual results and combined results to have a better understanding of SQL injection detection causes. In the individual detection results, Elasticsearch and Logstash performed data parsing using filters and sent logs to Kibana for visualization and dashboard creation. Machine learning BayesNet results were displayed using BayesNet classifier. Apart from this, in combined results, the results depended on the order of applying these methodologies on log files.

CHAPTER 4

IMPLEMENTATION

This chapter describes the various modules across which this project was developed. It explains the detailed steps about the implementation and development of a secured Cloud system. This contains the overview of technology, a step-by-step procedure of various installations, and a description of some important code.

4.1 System Implementation summary

This summary section will describe the in-depth details of various functionalities implemented across the modules:

1. Client side implementation
    - Web Client UI Implementation
    - Log4j configuration
    - Deployment on Amazon EC2
2. Database Implementation
3. Log analysis using Weka
4. Log analysis using Pattern Matching system

Each section of the implementation section explains the important functionality of every module, its characteristics, and screenshots of the code in detail.

4.2 Client-side implementation

4.2.1 Web Client UI implementation

In this module implementation, a social networking website was created for employees in a team. Information sharing is a key to success for effective communication in a team and this information may be confidential. The users of the web application were the admin and authorized employees. First, the admin authorized the employees who should have access to the web application. Second, only authorized employees could register into the web application with the details given by the admin for authorization such as employeeid, firstname, lastname. Third, once an employee logged in to the website, they could create a post with the content related to the company, comment on previous posts created, and delete posts created previously. I used Java, Bootstrap, HTML, CSS, and JavaScript to build this web application.

4.2.2 Log4j configuration

Log4j is a logging framework written in Java. Log4j comprises of three components: Loggers, appenders, and Layouts, as discussed in Chapter 2. In this project, log4j was used to log employee activities into the web application. The workflow of logging for this project is to first initiate a logger and send some text statements with the appropriate log levels to it. Before proceeding further, logger first checks the configuration file and sees if it has to print this log level or ignore it. Second, the logger checks which appenders are associated with them and redirects this logging statement to each appender. Last, when appenders receive logging statements, they check the configuration file about the layout format, and based on the specified format, the logging statement is printed in the specified appender.

Step1: Download and Add Log4j jar file in the project

In order to add the logging functionality to the project using Log4j, download the Log4j jar file and unzip it in a folder. In Eclipse, right-click the project and select Properties. In Properties, select Java Build Path > Libraries Tab > Add Jars > browse jars > add log4j-1.2.17.jar

Step2: Add logging functionality in a Java class

For the simple Java class, first add a log4j import statement in the class definition:

```
import org.apache.log4j.Logger;
```

Second, create the variable for logger and initiate it with the reference to the root-level logger. Add the below statement outside the main method, but in class definition only:

```
static final Logger logger = Logger.getRootLogger();
```

Last, initiate the logging statements using a variable called logger variable. Here, logging levels like WARN, INFO, ERROR, DEBUG and FATAL are considered and used according to the logging statement:

```
logger.debug("debug message");
logger.info("info message");
logger.warn("warn message");
logger.error("error message");
logger.fatal("fatal message");
```

Step3: Add Log4j Properties file

There are two ways to configure log4j: use the properties file of log4j or configuration in XML format. For this project, the log4j properties file is used for configuration. To initiate the configuration, first place the log4j properties file in the src folder of the web application. The same properties file can be used for any Java class which has the logging capabilities.

Here is the Log4j.properties file for this project:

```
# set the root logger to DEBUG.
log4j.rootLogger=DEBUG, LOGWEB

# MonitorLog - used to log messages
log4j.appender.LOGWEB=org.apache.log4j.RollingFileAppender
log4j.appender.LOGWEB.File=${catalina.base}/logs/appfinal1.txt
log4j.appender.LOGWEB.layout=org.apache.log4j.PatternLayout
log4j.appender.LOGWEB.layout.ConversionPattern="%d{ISO8601}"\t%p\t"
%X{RemoteAddress}"\t"%X{url}"\t"%X{user}"\t"%X{sessionid}"\t"%m"\t?
%n
```

```
log4j.logger.*=DEBUG,LOGFILE,LOGAPP,LOGWEB
```

In the above properties file, the first line describes the setup of the log level of the logger to DEBUG and after the comma in the same line, it tells the logger the name of the appender called LOGWEB, or whatever it's named. In the second line, this configures the specified appender name and sets it to RollingFileAppender which is used to print log statements in a file. Also, the path is specified to access the logging file used for logging statement. Besides logging the statements, another important functionality is formatting and to achieve this, the code adds the layout to the configuration file. Once all the steps are followed, all employee activities are logged each time an event is performed on the website.

4.2.3 Deployment on Amazon EC2

Steps to follow for deploying web application:

Step 1: Launch Amazon EC2 instance

A micro-single Amazon EC2 instance is created. While launching an instance, there are two mandatory things: choosing a security group and a key pair to enable a Linux instance through SSH. Before trying to connect to an instance from any Linux instance, we also need to get the public DNS name of the instance using the Amazon EC2 console.

Step 2: Export project from Eclipse IDE

To export an entire project as a war file by following these steps:

Select File > Export > War Export

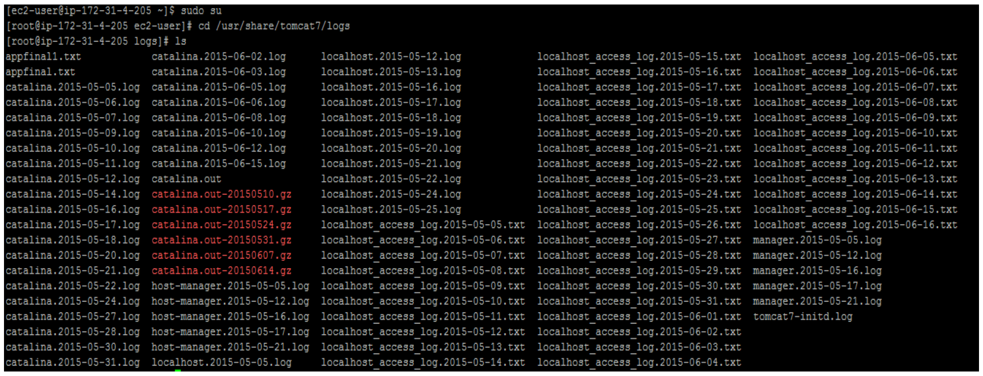
Step 3: File transfer through WinSCP

Download WinSCP and create a new session to connect to the Amazon EC2 server. To create a session, we need to include the hostname, username, and password, and in the advanced section of the session, add private key pair file. Once a user logs in to the WinSCP, the exported war file is transferred from a local system to remote server '/home/ec2-user'.

Step 4: Putty for Linux instance

Putty is used to connect to our Linux instance for deploying the project war file. For the instance connection, the user needs to create a session by inserting information in specified fields like hostname and port number, and the user also needs to upload SSH authentication private .ppk file. Here, the username for session login is ec2-user. Once the session login is successful, download tomcat7 and configure it in the Linux instance. Finally, to deploy the web application, copy the war file from '/home/ec2-user' path, which has been copied previously to the tomcat server path '/usr/share/tomcat7/webapps'.

Whenever employees perform some activity on the Cloud application, each event is logged in a log file mentioned in the log4j properties file, and placed on '/usr/share/tomcat7/logs' path on the Linux instance



```
[ec2-user@ip-172-31-4-205 ~]$ sudo su
[root@ip-172-31-4-205 ec2-user]# cd /usr/share/tomcat7/logs
[root@ip-172-31-4-205 logs]# ls
appfinal1.txt            catalina.2015-06-02.log      localhost.2015-05-12.log             localhost_access_log.2015-05-15.txt  localhost_access_log.2015-06-05.txt
appfinal.txt             catalina.2015-06-03.log      localhost.2015-05-13.log             localhost_access_log.2015-05-16.txt  localhost_access_log.2015-06-06.txt
catalina.2015-05-05.log  catalina.2015-06-05.log      localhost.2015-05-16.log             localhost_access_log.2015-05-17.txt  localhost_access_log.2015-06-07.txt
catalina.2015-05-06.log  catalina.2015-06-06.log      localhost.2015-05-17.log             localhost_access_log.2015-05-18.txt  localhost_access_log.2015-06-08.txt
catalina.2015-05-07.log  catalina.2015-06-08.log      localhost.2015-05-18.log             localhost_access_log.2015-05-19.txt  localhost_access_log.2015-06-09.txt
catalina.2015-05-09.log  catalina.2015-06-10.log      localhost.2015-05-19.log             localhost_access_log.2015-05-20.txt  localhost_access_log.2015-06-10.txt
catalina.2015-05-10.log  catalina.2015-06-12.log      localhost.2015-05-20.log             localhost_access_log.2015-05-21.txt  localhost_access_log.2015-06-11.txt
catalina.2015-05-11.log  catalina.2015-06-15.log      localhost.2015-05-21.log             localhost_access_log.2015-05-22.txt  localhost_access_log.2015-06-12.txt
catalina.2015-05-12.log  catalina.out                 localhost.2015-05-22.log             localhost_access_log.2015-05-23.txt  localhost_access_log.2015-06-13.txt
catalina.2015-05-14.log  catalina.out-20150510.gz     localhost.2015-05-24.log             localhost_access_log.2015-05-24.txt  localhost_access_log.2015-06-14.txt
catalina.2015-05-16.log  catalina.out-20150517.gz     localhost.2015-05-25.log             localhost_access_log.2015-05-25.txt  localhost_access_log.2015-06-15.txt
catalina.2015-05-17.log  catalina.out-20150524.gz     localhost_access_log.2015-05-05.txt  localhost_access_log.2015-05-26.txt  localhost_access_log.2015-06-16.txt
catalina.2015-05-18.log  catalina.out-20150531.gz     localhost_access_log.2015-05-06.txt  localhost_access_log.2015-05-27.txt  manager.2015-05-05.log
catalina.2015-05-20.log  catalina.out-20150607.gz     localhost_access_log.2015-05-07.txt  localhost_access_log.2015-05-28.txt  manager.2015-05-12.log
catalina.2015-05-21.log  catalina.out-20150614.gz     localhost_access_log.2015-05-08.txt  localhost_access_log.2015-05-29.txt  manager.2015-05-16.log
catalina.2015-05-22.log  host-manager.2015-05-05.log  localhost_access_log.2015-05-09.txt  localhost_access_log.2015-05-30.txt  manager.2015-05-17.log
catalina.2015-05-24.log  host-manager.2015-05-12.log  localhost_access_log.2015-05-10.txt  localhost_access_log.2015-05-31.txt  manager.2015-05-21.log
catalina.2015-05-27.log  host-manager.2015-05-16.log  localhost_access_log.2015-05-11.txt  localhost_access_log.2015-06-01.txt  tomcat7-initd.log
catalina.2015-05-28.log  host-manager.2015-05-17.log  localhost_access_log.2015-05-12.txt  localhost_access_log.2015-06-02.txt
catalina.2015-05-30.log  host-manager.2015-05-21.log  localhost_access_log.2015-05-13.txt  localhost_access_log.2015-06-03.txt
catalina.2015-05-31.log  localhost.2015-05-05.log     localhost_access_log.2015-05-14.txt  localhost_access_log.2015-06-04.txt
```

Figure 3: Path specified for log files on the Linux instance

To check whether the deployment is successful or not, go to '/usr/share/tomcat7/logs' and check the log file with the logging statements.

4.3 Database Implementation

The database is implemented on MYSQL. MYSQL is an open-source database system which runs on the server and used on web applications. The data in this database is stored in the form of tables. In this project, the database was used to store the personal details and activities of employees. The detailed steps for connecting the database from the Cloud application are given below:

Step 1: Download MYSQL workbench GUI for the users to create a database and perform MYSQL queries on the top of a database.

Step 2: Open Linux instance and download MYSQL on the top of it.

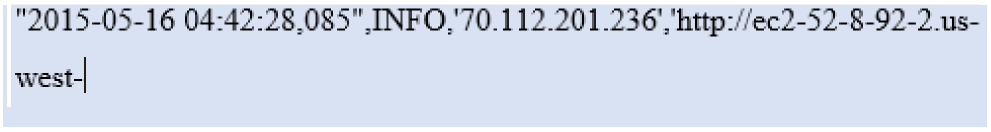
Step 3: Create a MYSQL connection to connect to a remote server by providing details such as SSH hostname, name of the SSH user to connect to remote server, and SSH key file (i.e. .pem file); also, specify the MYSQL local instance information such as hostname, port number (which is, by default, 3306), and password for the MYSQL database. The database has five tables: adminlogin, authorizedemp, empcomment, empinfo, and emppost.

4.4 Log analysis using Weka

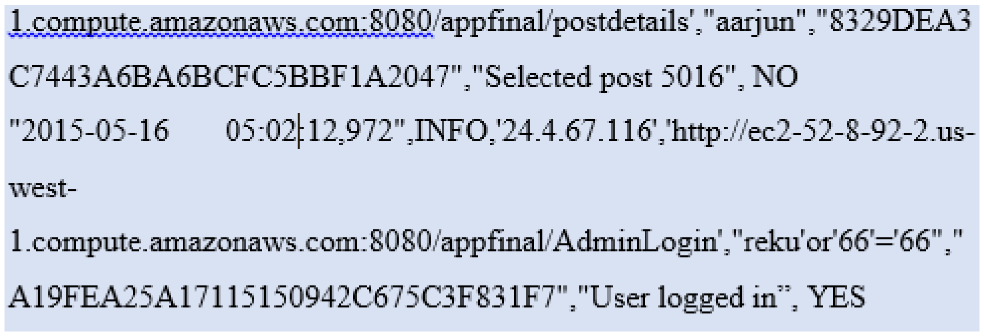
4.4.1 Data Collection

In Classification, the user provides a set of data which is logged from the Cloud application, and uses a part of it to train the classifier model and then predict the result of the remaining data. The dataset used for learning is called the trained dataset. Using the learning algorithm, a model is learned from the training data, a set of test data used to evaluate and assess the model accuracy. Supervised Learning is the task of learning a classification model using the known dataset that can be used to classify new. In detail, to train the model, each instance is provided with a labeled class which is helpful in predicting the correct class for unclassified future datasets. In this project, we consider classing a YES or NO. Here, YES is considered a malicious instance and NO a regular instance.

Below is the sample data with the class labels as YES or NO:



```
"2015-05-16 04:42:28,085",INFO,'70.112.201.236','http://ec2-52-8-92-2.us-
west-
```



```
1.compute.amazonaws.com:8080/appfinal/postdetails',"aarjun","8329DEA3
C7443A6BA6BCFC5BBF1A2047","Selected post 5016", NO
"2015-05-16      05:02:12,972",INFO,'24.4.67.116','http://ec2-52-8-92-2.us-
west-
1.compute.amazonaws.com:8080/appfinal/AdminLogin',"reku'or'66'='66","
A19FEA25A17115150942C675C3F831F7","User logged in", YES
```

As the log file is large, data cleaning should be done to the log file so that it can be easily loadable in Weka for further processing.

4.4.2 Data Pre-processing

There are three steps for data pre-processing in Weka:

a. Converting a Text file to ARFF file format

Before loading the data into Weka, convert the log file format from txt file to ARFF file format. ARFF file format consists of two sections: header section and data section. The header section consists of relation name and a set of attributes with their datatypes**.** In addition to ARFF data file format, WEKA also has the capability of reading the .csv format.

Filter attributes and Feature Selection

We can choose the filters from a filter panel in Weka. In this project, String to work vector filter is used. Once the filter is applied, the immediate results can be stored as a separate data file and it treats each step as a separate session. Each session can, in turn, save as an ARFF file using the Save button. In data pre-processing, feature selection is very important in improving the accuracy. To achieve better accuracy, select features which are meaningful for classification and useful for further predictions on testing data.

b. Methods for improving accuracy

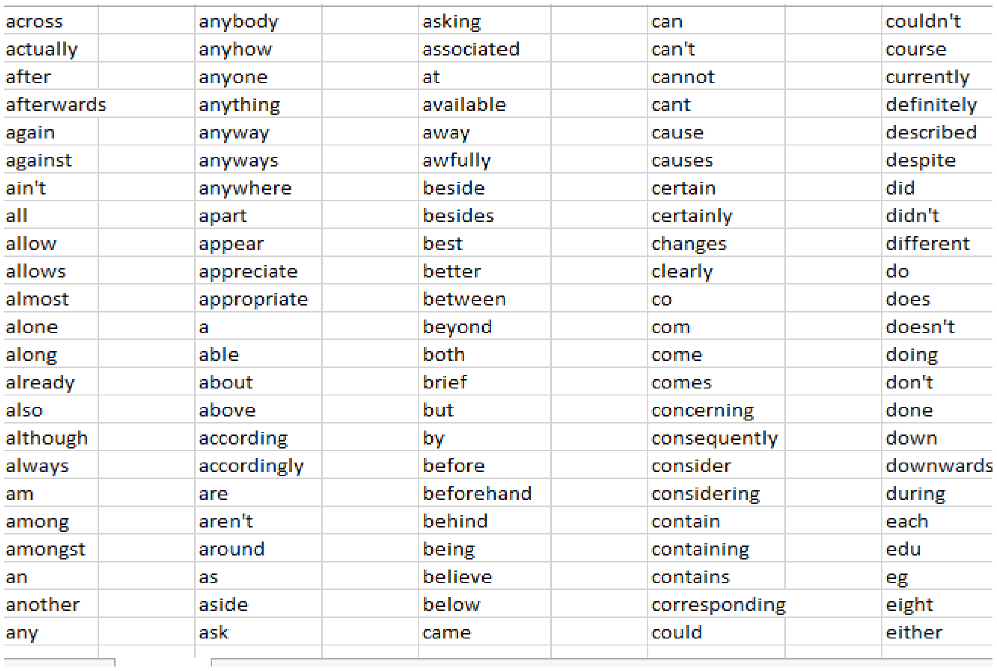
This section mainly focuses on selecting the features and also various methods for improving accuracy.

Method1: Removing Unnecessary features

Removing unnecessary features reduces space and also improves the classifier accuracy. To achieve this, we should select features which help further classification. In this project, timestamp and session id feature are not helpful and, therefore, are removed from the feature list.

Method2: Stop word list

Stop words are a set of most commonly-used single-words in a language. General examples of stop words are: ‘the’, ‘a’, ‘an’, and ‘above’. Accuracy can be improved by adding a custom stop word list. Comments in this file are ignored.



across	anybody	asking	can	couldn't
actually	anyhow	associated	can't	course
after	anyone	at	cannot	currently
afterwards	anything	available	cant	definitely
again	anyway	away	cause	described
against	anyways	awfully	causes	despite
ain't	anywhere	beside	certain	did
all	apart	besides	certainly	didn't
allow	appear	best	changes	different
allows	appreciate	better	clearly	do
almost	appropriate	between	co	does
alone	a	beyond	com	doesn't
along	able	both	come	doing
already	about	brief	comes	don't
also	above	but	concerning	done
although	according	by	consequently	down
always	accordingly	before	consider	downwards
am	are	beforehand	considering	during
among	aren't	behind	contain	each
amongst	around	being	containing	edu
an	as	believe	contains	eg
another	aside	below	corresponding	eight
any	ask	came	could	either

Figure 4: Sample list of stop words

To set the custom stop word list, first choose the filter for filtering the attributes and any filter that provides you with the option of a stop word list. filters.unsupervised.attribute.StringToWordVector is the filter used in this project to provide you with the custom stop word list option, and to load the external stop word file. The custom stop word list file format is to have one stop word per line. For this project, a custom stop word list is created with words list like ‘and’, ‘or’, ‘from’ etc.

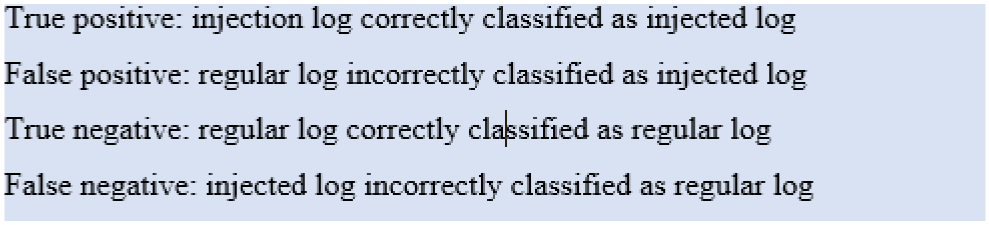
4.4.3 Cross Validation

Cross Validation is a technique used to evaluate an algorithm by performing the different splits of the data; the results are averaged over the each split result. A five-fold cross-validation was used for this project. With five-fold cross validation, the entire dataset is divided to make four pieces for training and one piece for testing. The different data segments are taken to perform five-fold cross validation using the same method. Once cross validation is finished, for the sixth time, Weka runs the algorithm on the entire dataset and produces a classifier. Weka, by default, uses stratified cross validation (i.e. when the initial division into five parts is performed on the dataset); each class should have the correct amount of each of the class values.

4.4.4 Data Classification

Once the data pre-processing is performed, we select a classify tab where a machine learning algorithm is selected. In this project, BayesNet algorithm was used for data classification. A select dataset was used for training in order to build a model in Weka. The result produces the time that was taken to build the model, correctly classified and incorrectly classified instances, and detailed accuracy by class and confusion matrix.

Here, the confusion matrix tells how good the model is working with regards to correctly and incorrectly classified instances. Below is the description about confusion matrix:


True positive: injection log correctly classified as injected log

False positive: regular log incorrectly classified as injected log

True negative: regular log correctly classified as regular log

False negative: injected log incorrectly classified as regular log

For future predictions, we save the build classification model. In order to validate the classification model, we consider the test set, use the saved model to run test set, and assess the accuracy. To perform this, we have to load the test set that was not present in the training set we used to create a model. Then, with the help of the created model, the test set is checked and gives us the details about the performance of the model. Finally, we compare the correctly classified instances in the test set with those in the training set, to see the accuracy of the model. In regards to the gained accuracy, we can decide that this model can be applied for future predictions on unknown or future data.

4.5 Log analysis using Pattern matching system

4.5.4 Logstash, Elasticsearch and Kibana setup

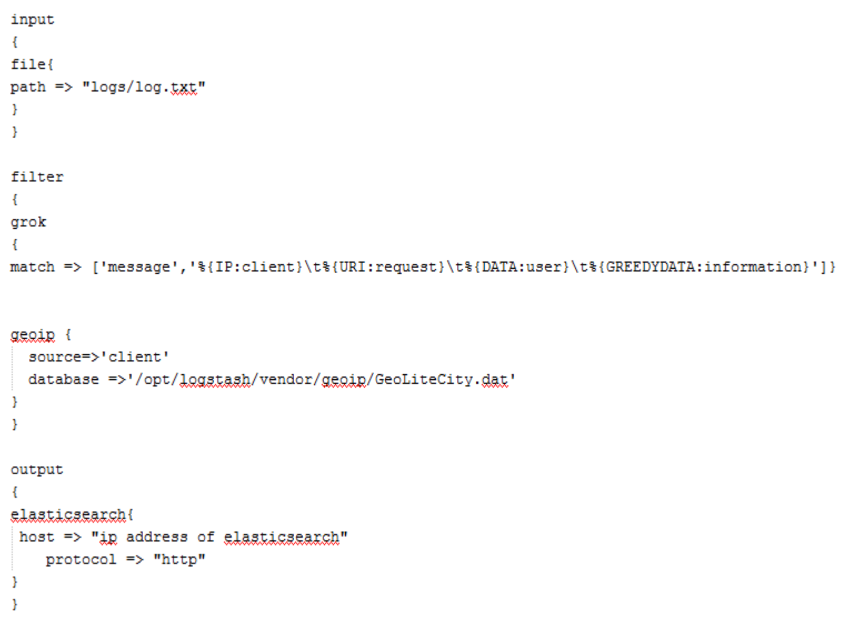
To set up a pattern-matching system, the following steps are performed:

Step1: Installation

Download Logstash, Elasticsearch and Kibana from the appropriate website links, and install on the Linux instance created in the previous section [22].

Step2: Configure Logstash and store logs in Elasticsearch

The below configuration file is placed in '/etc/logstash/config' folder. This file has three sections: input section, filter section, and output section. In the first section of the file, the log file of the incoming path is specified. The second section of the file is used for the configuration of the filter. Before forwarding the logs to Elasticsearch for storing, Logstash performs parsing of logs using a Grok filter. The below Grok filter takes the logs related to IP address, URI, user, and information, and considers them as separate columns; any remaining data is placed as a single column in Elasticsearch.



```
input
{
file{
path => "logs/log.txt"
}
}

filter
{
grok
{
match => ['message','%{IP:client}\t%{URI:request}\t%{DATA:user}\t%{GREEDYDATA:information}']}


geoip {
  source=>'client'
  database =>'/opt/logstash/vendor/geoip/GeoLiteCity.dat'
}
}

output
{
elasticsearch{
 host => "ip address of elasticsearch"
    protocol => "http"
}
}
```

Figure 5: Logstash Configuration File

Logstash has a geoip file called GeoLiteCity.dat for detecting the locations of client IP addresses. The third section of the file is for output configuration, which defines the location where the filtered logs get stored in Elasticsearch.

Step3: Configure Kibana

To use Kibana, configure at least one index pattern where we have to map Logstash index. Select timestamp field and it will redirect to Kibana main page.

4.5.5 Data Analysis and Visualization

Kibana is an analytics and visualization tool that builds on the top of Elasticsearch to have a better insight of data. In this project, we used Kibana 4 to filter and visualize data. When we first connect to Kibana 4, a discover tab is shown with the logs that we recently received. However, based on a search query written in search tab, we can filter this stream logs and also restrict this search for the specific time by using Time filters. In this project, we used Lucene query language for pattern recognition. To achieve this, two Lucene queries were written to detect the SQL injection logs and non-SQL injection logs.

Here is the Lucene query for detecting SQL injection attacks:

```
(((select || delete) && from) || (update && set) || (union && select && from) ||
((Create || (alter && (add || drop || modify))) && (table || database))) || (insert &&
into && values) || (drop && (table || database || index))  || *join || user:(or || and)
```

The query syntax is self-evident and allows wildcards, boolean operators, and other group and field filtering to match the patterns.

Here is the Lucene query for filtering non-SQL injection attacks:

```
!((((select || delete) && from) || (update && set) || (union && select && from) ||
((Create || (alter && (add || drop || modify))) && (table || database))) || (insert && into
&& values) || (drop && (table || database || index))  || *join || user:(or || and))
```

These two queries are saved for further use and can be opened at any time by clicking the loaded save search. Also, the saved searches are used when creating visualizations and dashboards.

Kibana Visualize page : The Kibana Visualize page allows users to create, edit, and share visualizations. Below are the types of visualizations created for this project:

i. A data table visualization provides a detailed breakdown of SQL injection and non SQL injection logs.

ii. The pie charts display the top-most IP addresses that perform SQL injection attack with its top most URLs.

iii. The title map visualizations detect the locations of malicious users of the web applications. Furthermore, a zoomed version of locations is implemented for further analysis.

iv. The vertical bar charts show the time frames of the log event occurrences.

Kibana Dashboard

With a dashboard, multiple visualizations created are combined onto a single page, and these can be further filtered with the help of saved Lucene queries.

To create a dashboard, follow these steps:

a. Add multiple saved visualizations on a blank dashboard.
b. Rearrange the visualizations based on SQL injection and non-SQL injection categories.
c. Save the dashboard for new, incoming data.

Kibana Export

To obtain the export option, a dashboard with visualizations is created to help security analysts download raw data for doing further investigations.

CHAPTER 5

EXPERIMENT AND RESULTS

The experiment chapter describes the details of the dataset used in the project. It explains the steps used to execute a secured Cloud system.

5.1 The experiment data

The dataset was made up of web application logs generated using the log4j framework. The total number of logs used for the experiment is 10,000. The data file is in both .txt and ARFF formats. The log files contain these fields: timestamp, log level, IP address, session id, URL, username, and message. All of the data was cleaned and filtered according to the requirements in the data pre-processing stage.

5.2 BayesNet Classifier using Weka

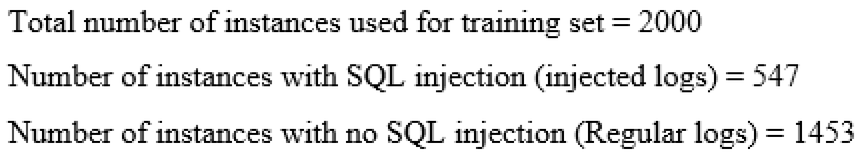
To experiment on the log analysis using BayesNet Classifier, an open-source tool called Weka was used. This tool has in-built algorithms to train a model using a training set, and it also predicts the accuracy of test data.

5.2.1 Execution steps

Step 1:

The Weka tool uses ARFF file format to process and analyze data. To achieve this, convert the text file format into ARFF format, add @ symbol to header section and data attribute section separately, and use a comma as the delimiter for the data. The training set input file is formatted and converted .to arff format.

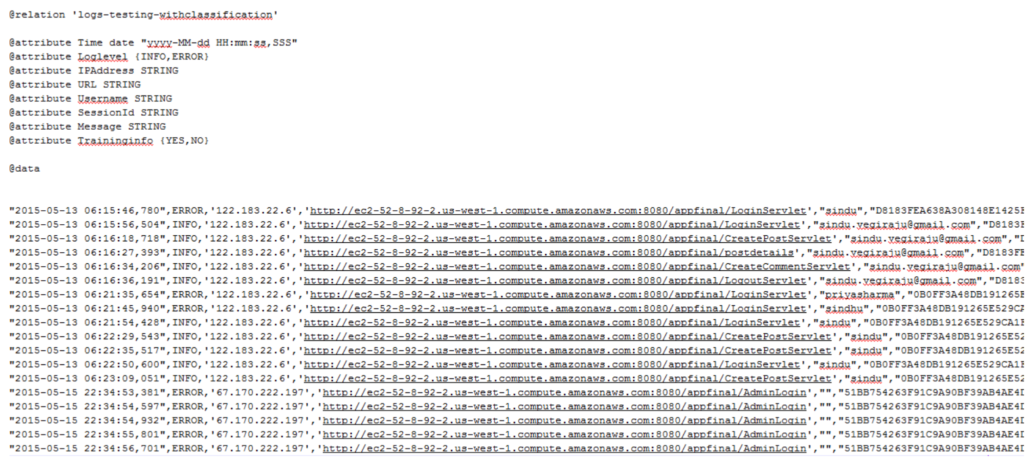
Training set details



Total number of instances used for training set = 2000

Number of instances with SQL injection (injected logs) = 547

Number of instances with no SQL injection (Regular logs) = 1453



```
@relation 'logs-testing-withclassification'

@attribute Time date "yyyy-MM-dd HH:mm:ss,SSS"
@attribute Loglevel {INFO,ERROR}
@attribute IPAddress STRING
@attribute URL STRING
@attribute Username STRING
@attribute SessionId STRING
@attribute Message STRING
@attribute Traininginfo {YES,NO}

@data


"2015-05-13 06:15:46,780",ERROR,'122.183.22.6','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/LoginServlet',"sindu","D8183FEA638A308148E1425
"2015-05-13 06:15:56,504",INFO,'122.183.22.6','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/LoginServlet',"sindu.vegiraju@gmail.com","D8183
"2015-05-13 06:16:18,718",INFO,'122.183.22.6','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/CreatePostServlet',"sindu.vegiraju@gmail.com","
"2015-05-13 06:16:27,393",INFO,'122.183.22.6','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/postdetails',"sindu.vegiraju@gmail.com","D8183F
"2015-05-13 06:16:34,206",INFO,'122.183.22.6','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/CreateCommentServlet',"sindu.vegiraju@gmail.com"
"2015-05-13 06:16:36,191",INFO,'122.183.22.6','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/LogoutServlet',"sindu.vegiraju@gmail.com","D8183
"2015-05-13 06:21:35,654",ERROR,'122.183.22.6','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/LoginServlet',"priyasharma","0B0FF3A48DB191265
"2015-05-13 06:21:45,940",ERROR,'122.183.22.6','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/LoginServlet',"sindhu","0B0FF3A48DB191265E529C
"2015-05-13 06:21:54,428",INFO,'122.183.22.6','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/LoginServlet',"sindu","0B0FF3A48DB191265E529CA1
"2015-05-13 06:22:29,543",INFO,'122.183.22.6','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/CreatePostServlet',"sindu","0B0FF3A48DB191265E52
"2015-05-13 06:22:35,517",INFO,'122.183.22.6','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/CreatePostServlet',"sindu","0B0FF3A48DB191265E52
"2015-05-13 06:22:50,600",INFO,'122.183.22.6','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/LoginServlet',"sindu","0B0FF3A48DB191265E529CA1
"2015-05-13 06:23:09,051",INFO,'122.183.22.6','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/CreatePostServlet',"sindu","0B0FF3A48DB191265E52
"2015-05-15 22:34:53,381",ERROR,'67.170.222.197','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/AdminLogin',"","51BB754263F91C9A90BF39AB4AE4
"2015-05-15 22:34:54,597",ERROR,'67.170.222.197','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/AdminLogin',"","51BB754263F91C9A90BF39AB4AE4
"2015-05-15 22:34:54,932",ERROR,'67.170.222.197','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/AdminLogin',"","51BB754263F91C9A90BF39AB4AE4
"2015-05-15 22:34:55,801",ERROR,'67.170.222.197','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/AdminLogin',"","51BB754263F91C9A90BF39AB4AE4
"2015-05-15 22:34:56,701",ERROR,'67.170.222.197','http://ec2-52-8-92-2.us-west-1.compute.amazonaws.com:8080/appfinal/AdminLogin',"","51BB754263F91C9A90BF39AB4AE4
```

Figure 6: Sample log file in ARFF format

Step 2:

The training set consists of 2,000 log instances. Once the training set is loaded, in the feature selection, tasks in the pre-processing tab are applied to increase the classification performance. To improve the classification accuracy, we remove the features which are not considered important such as timestamp and session id. Select StringtoWord vector filter to convert string attributes into a set of attributes that represent the occurrences of words from text present in the strings. Upload the stop world list file in Weka to improve accuracy.

Step 3:

Now it's time to choose a machine learning algorithm and make predictions. In this project, a BayesNet model is created using training set, fivefold cross validation, string to word vector filter and custom stop word list. The trained model is saved to the local system.

	1	2	3	4	5
Detection	78.4%	78.5%	79%	78.6%	79.5%

Figure: BayesNet Cross Validation Result

Mean detection rate: 78.80% and Standard deviation: 0.91%

Step 4

Now, the saved trained model is used on the test data to give classification for unclassified instances.

Test set details

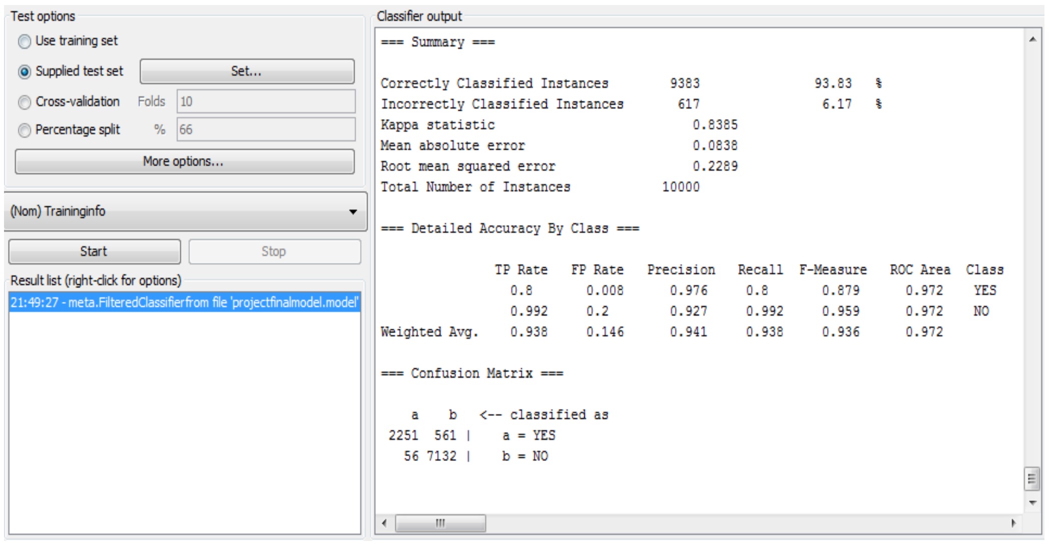
Total number of instances=10000

Number of instances with SQL injection (injected logs) = 2812

Number of instances with no SQL injection (Regular logs) = 7188

5.2.2 Results

Test Set Result Obtained



```
=== Summary ===

Correctly Classified Instances        9383               93.83   %
Incorrectly Classified Instances       617                6.17   %
Kappa statistic                          0.8385
Mean absolute error                      0.0838
Root mean squared error                  0.2289
Total Number of Instances            10000

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0.8       0.008      0.976     0.8       0.879      0.972    YES
                 0.992     0.2        0.927     0.992     0.959      0.972    NO
Weighted Avg.    0.938     0.146      0.941     0.938     0.936      0.972

=== Confusion Matrix ===

    a    b   <-- classified as
 2251  561 |    a = YES
   56 7132 |    b = NO
```

Figure 7: Results of Test set

True Positive(YES)	False Positive(YES)	True Negative(NO)	False Negative(NO)
2251	561	7132	56

Accuracy obtained for BayesNet Classifier using Weka for the given data set

Percentage of correctly classified total instances = 93.83%

Percentage of correctly classified SQL instances = 80.05%

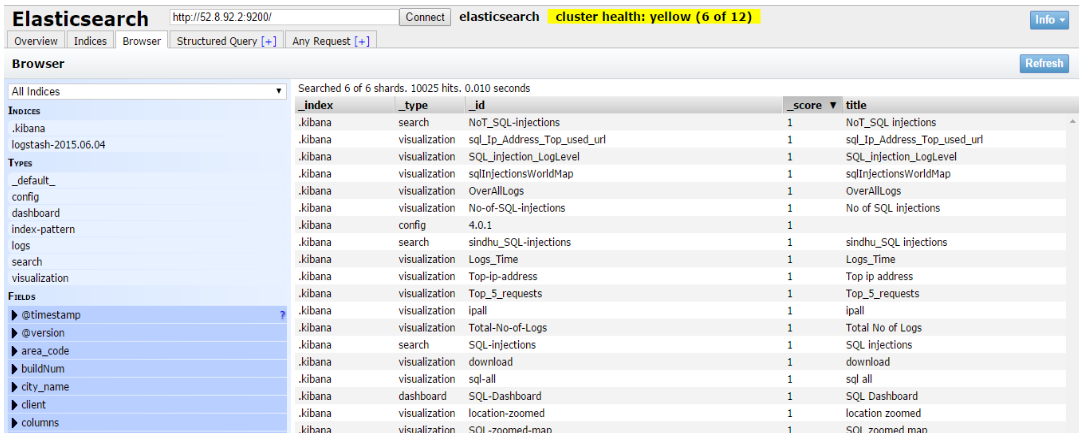
5.3 Pattern Matching System

To perform log analysis using pattern matching system for logs, we used three tools: Logstash for data collection, Elasticsearch for search and analysis, and Kibana for data visualization.

5.3.1 Execution steps

Step 1: Place the path of the log file in Logstash configuration file to get the stream data from the Cloud. This Logstash configuration file provides data parsing on the dataset before sending it to Elasticsearch.

Step 2: Check whether a log file is stored in Elasticsearch or not by opening the Elasticsearch data store in the Elasticsearch web browser.



Searched 6 of 6 shards. 10025 hits. 0.010 seconds

_index	_type	_id	_score ▼	title
.kibana	search	NoT_SQL-injections	1	NoT_SQL injections
.kibana	visualization	sql_Ip_Address_Top_used_url	1	sql_Ip_Address_Top_used_url
.kibana	visualization	SQL_injection_LogLevel	1	SQL_injection_LogLevel
.kibana	visualization	sqlInjectionsWorldMap	1	sqlInjectionsWorldMap
.kibana	visualization	OverAllLogs	1	OverAllLogs
.kibana	visualization	No-of-SQL-injections	1	No of SQL injections
.kibana	config	4.0.1	1	
.kibana	search	sindhu_SQL-injections	1	sindhu_SQL injections
.kibana	visualization	Logs_Time	1	Logs_Time
.kibana	visualization	Top-ip-address	1	Top ip address
.kibana	visualization	Top_5_requests	1	Top_5_requests
.kibana	visualization	ipall	1	ipall
.kibana	visualization	Total-No-of-Logs	1	Total No of Logs
.kibana	search	SQL-injections	1	SQL injections
.kibana	visualization	download	1	download
.kibana	visualization	sql-all	1	sql all
.kibana	dashboard	SQL-Dashboard	1	SQL Dashboard
.kibana	visualization	location-zoomed	1	location zoomed
.kibana	visualization	SQL-zoomed-map	1	SQL zoomed map

Figure 8: Elasticsearch web user interface

Step 3: Open the Kibana web browser by providing the URL of the server. Before navigating to the main page of Kibana, select timestamp to create the index. After selecting the index, the discover page in Kibana is displayed with the timestamp of the most recent log file added.

Step 4: Open the visualize tab in Kibana, then create different visualizations by using the loaded query written to detect SQL injection attacks.

Step 5: Count of the number of SQL injections shown using the data table visualization filter, with the help of the SQL injection detection saved query search.

Step 6: Create a tile map visualization to detect the locations of malicious users and authorized users. To achieve this, select the geo IP address field and geo hash aggregation while creating a visualization.

Step 7: Create a pie chart type of visualization to detect the top-most malicious IP address with most commonly used URLs.

Step 8: Create a dashboard with the saved visualizations and rearrange them according to the requirements.

5.3.2 Results

Accuracy obtained for Pattern matching System for the given data set

Percentage of correctly classified instances: 85.3%

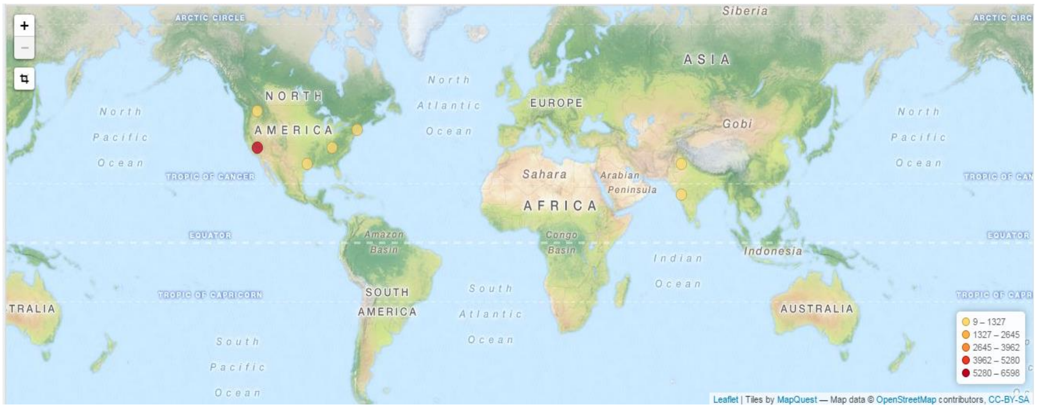


Figure 9: A tile map showing the locations of all IP address who performed activity on cloud application

5.4 BayesNet Classifier to Pattern matching system

This methodology is demonstrated by using the BayesNet output and sending it to the Pattern matching system as an input for further detection; this improves accuracy better than with the individual detection results.

5.4.1 Execution steps

Step 1: Download BayesNet result from Weka tool.

Step 2: Use the following UNIX scripts to convert the BayesNet output file into the format supported by Kibana:

UNIX script used to replace commas with tabs

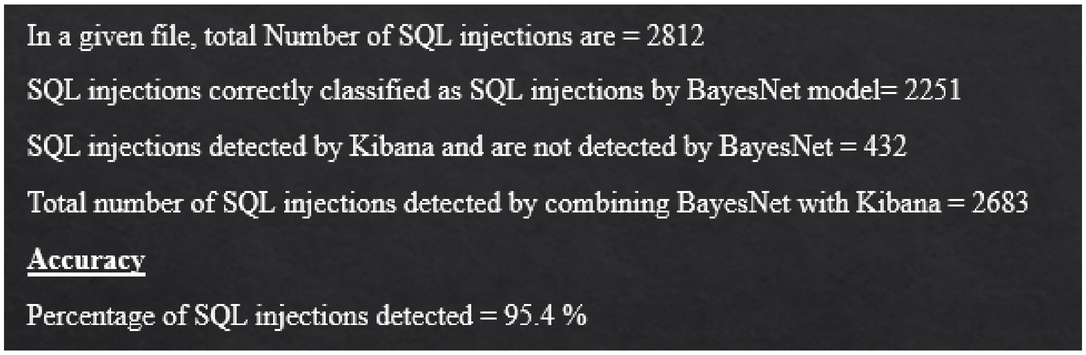
```
sed 's/,/\t/g' result.csv > newresult.csv
```

UNIX script used to delete header part of a file

```
sed -i 1,16d newresult.csv
```

Step 3: As output files don't have the timestamp feature, we infer timestamp as current loading timestamp for loading the output file.

5.4.2 Results



In a given file, total Number of SQL injections are = 2812

SQL injections correctly classified as SQL injections by BayesNet model= 2251

SQL injections detected by Kibana and are not detected by BayesNet = 432

Total number of SQL injections detected by combining BayesNet with Kibana = 2683

<u>Accuracy</u>

Percentage of SQL injections detected = 95.4 %

Figure 10: BayesNet to Pattern Matching System Result

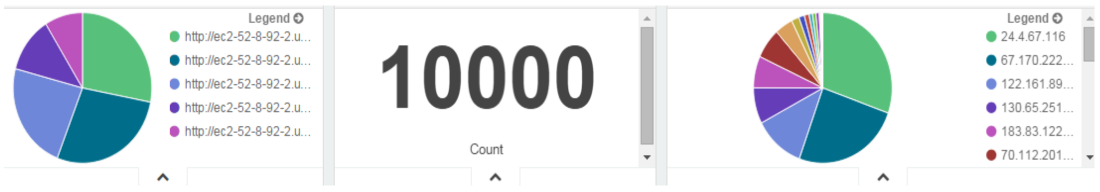

Legend
http://ec2-52-8-92-2.u...
http://ec2-52-8-92-2.u...
http://ec2-52-8-92-2.u...
http://ec2-52-8-92-2.u...
http://ec2-52-8-92-2.u...
10000
Count
Legend
24.4.67.116
67.170.222...
122.161.89...
130.65.251...
183.83.122...
70.112.201...

Figure 11: A dashboard demonstrating the count of dataset and visualizations of top URLs and IP addresses of all the web application users.

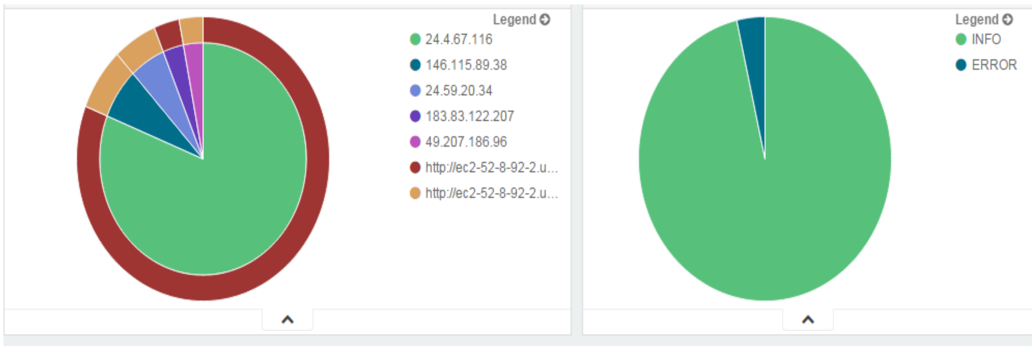


Figure 12: Pie chart illustrating the top IP address of SQL injection attack with top most used URL's and a pie chart showing the log level of the instances

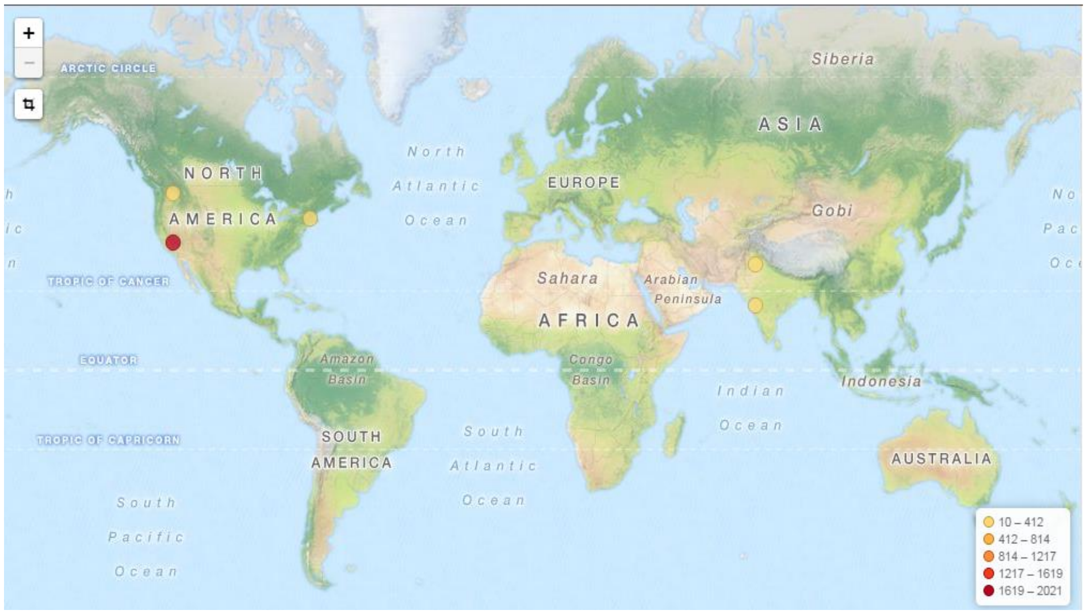


Figure 13: A tile map showing the locations of all IP addresses which performed SQL injection attack on a Cloud application

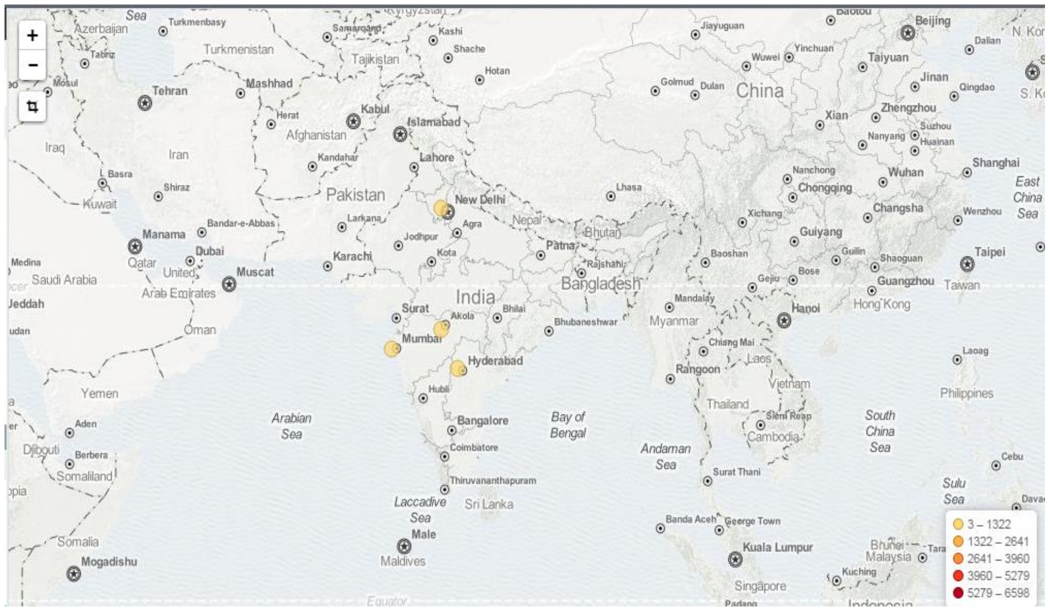


Figure 14: Zoomed version of IP address locations which performed SQL injection attacks.

5.5 Pattern matching system to BayesNet Classifier

This methodology is demonstrated by using the Kibana non-detected logs and sending it to the BayesNet as an input for further detection, which improves accuracy better than with the individual detection results.

5.5.1 Execution steps

Step 1: Create a visualization in Kibana which illustrates all the logs that are apart from SQL injection attacks.

Step 2: Kibana provides the export option to share or download raw data from the selected visualization. Here, we exported a CSV file from the created visualization.

Step 3: Use this UNIX script to add the class field to the CSV file.

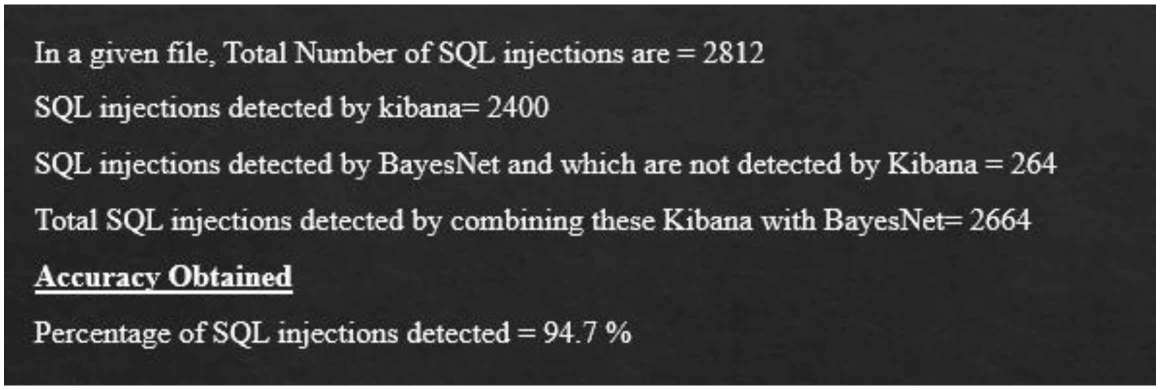
```
sed "s/\([0-9]\)\s*$/\?/" rawdata.csv > newdata.csv
```

Step 4: Use this UNIX script to convert CSV to ARFF file format.

```
cat newdata.csv >>new.arff
```

Step 5: Pre-process the data by removing the extra fields.

5.52 Results



```
In a given file, Total Number of SQL injections are = 2812
SQL injections detected by kibana= 2400
SQL injections detected by BayesNet and which are not detected by Kibana = 264
Total SQL injections detected by combining these Kibana with BayesNet= 2664
Accuracy Obtained
Percentage of SQL injections detected = 94.7 %
```

Figure 15: Pattern Matching System to BayesNet Result

CHAPTER 6

EVALUATION

This chapter describes comparisons between BayesNet (Machine Learning) and Kibana detection methodology. Furthermore, it describes why the accuracy increases when the analysis is performed with the combination of detection models.

Comparison between BayesNet (Machine Learning) and Kibana

BayesNet (Machine Learning)	***Kibana***
1. BayesNet is a learn-and-detect model.	1. This uses the Pattern Matching technique.
2. Data loading is performed manually by converting the txt or CSV file to ARFF file format.	2. Accepts live data streams and demonstrates live detection of SQL injections.
3. Training the model is required for future predictions on new data.	3. Training is not required for future predictions.
4. Manual classification of logs is performed for the training set. It takes time to pre-process data.	4. Manual classification is not performed on logs.
5. Queries are not required.	5. Lucene queries can be used to analyze the data.
6. Uses BayesNet classifier only and secured 80.05% (SQL injection logs) accuracy.	6. Uses Kibana only and secured 86% accuracy.

Reasons regarding SQL injections not detected in BayesNet and are detected by Kibana

i. There are some attributes such as log level and username used in SQL injection instances that may also contain regular instances in the training set. Based on this scenario, BayesNet classifier might not have detected SQL injection in some cases because this classifier reviews relations regardless of individual attributes.

ii. BayesNet could not detect some log instances because of new SQL injection patterns which are not contained in the training set. Also, these new SQL injection attack logs also contain other attributes, and these attributes belong to the regular logs in the training set. In this situation, Bayesnet classifier classifies this type of log as regular logs.

Example: drop table tablename

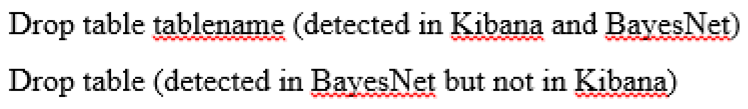
In the above example, ‘drop’ and ‘table’ fare requently used words that can be used in regular sentences, but the combination of these two regular words can be used to detect the SQL injection attack using Kibana query.

iii. BayesNet classifier could not detect some logs with the content of a large number of words belonging to the regular logs category. This is because BayesNet classifies the logs based on the probability of all the words present in a given log.

Reasons regarding SQL injections not detected in Kibana and are detected by BayesNet

i. Some of the logs not detected by Kibana contain special characters such as #, <,>, <=, >= in the username field. But BayesNet classifier also uses other attributes for classification and these logs are detected by BayesNet.

ii. In our logs, some of the SQL injection patterns only contain individual SQL keywords rather than combination. Kibana cannot identify such patterns because we used the grouping concept in the query to retrieve meaning information.



Drop table tablename (detected in Kibana and BayesNet)

Drop table (detected in BayesNet but not in Kibana)

CHAPTER 7

CONCLUSION AND FUTURE WORK

The main objective to provide security from SQL Injection attacks has thus been established. With numerous data flowing in everyday, it has become very important to focus on detecting SQL injection attacks that cause severe security problems on any web application hosted on the Cloud. The system was designed in such a way that it can expose the possible SQL injection attacks that an application is prone to. The implementation involves storing and maintenance of thousands of logs for every operation a user does on the application. On critical analysis of all these logs, it has always been a point to ensure that data security is provided to multiple users accessing a Cloud-based application.

For that purpose, two SQL injection detection methodologies were executed and their results have been evaluated to determine the root causes of the attacks. This information was forwarded to the Security Analyst to initiate proper actions against the attackers and resolve the problems accordingly. The two detection models, BayesNet classifier and Pattern Matching System, were implemented separately and also together on the stored log data. Results were compared and we concluded that, by applying these methodologies together as a combination, we detected a higher percentage of SQL injected log data when compared to applying them individually.

The accuracy percentage that resulted in detecting the SQL injection attacks on a Cloud-based application is 95.4 percent, if the order of applying models is from BayesNet to Pattern Matching. But if the models are applied vice-versa (from Pattern Matching to BayesNet), then the accuracy is 94.7 percent. Hence, the system successfully detected the SQL injection attacks on a Cloud application, ensuring an efficient way to provide end-to-end security.

Future Work

The goal to detect the SQL injection attacks on a Cloud-based application by analyzing the log files has been achieved to keep it more secure. As an extension to the system designed, further implementations will include analysis of large datasets and efficient evaluations of their results. Additionally, the results can be enhanced by implementing other feature extraction methodologies to achieve 100 percent accuracy in detecting the SQL injection attacks. This will help in detecting the attacks and resolving them effectively.

LIST OF REFERENCES

[1] R.R. Bouckaert, (2004). “Bayesian network classifiers in weka,” Department of Computer Science, University of Waikato.

[2] J. Cheng, & R. Greiner, (1999, July). “Comparing Bayesian network classifiers,” In Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence (pp. 101-108). Morgan Kaufmann Publishers Inc.

[3] G. Erik, (2014). “Introduction to Supervised Learning,” 1-2

[4] Feng Cheng, A. Azodi, D. Jaeger, C. Meinel, (2013). “Multi-Core Supported High Performance Security Analytics,” Dependable, Autonomic and Secure Computing (DASC), IEEE 11th International Conference.

[5] N.Friedman, D. Geiger, & M. Goldszmidt, (1997). “Bayesian network classifiers,” Machine learning, 29(2-3), 131-163.

[6] C. Gülcü, & S. Stark, (2003). “The complete log4j manual,” QOS. ch.

[7] G. Holmes, A. Donkin, & I.H. Witten, (1994, December). “Weka: A machine learning workbench,” In Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on (pp. 357-361). IEEE.

[8] Hussein Alnabulsi, Md Rafiqul Islam, Quazi Mamun. (2014). “Detecting SQL Injection Attacks Using SNORT IDS,” Computer Science and Engineering (APWC on CSE), Asia-Pacific World Congress, IEEE.

[9] T. Kalamatianos, K. Kontogiannis, P. Matthews, (2012). “Domain Independent Event Analysis for Log Data Reduction,” Computer Software and Applications Conference (COMPSAC), IEEE.

[10] B.R. Kandukuri, V.R. Paturi, & A. Rakshit, (2009, September). “Cloud security issues,” In Services Computing, 2009. SCC'09. IEEE International Conference on (pp. 517-520). IEEE.

[11] M. Kumar, M. Hanumanthappa, (2013). “Scalable Intrusion Detection Systems Log Analysis using Cloud Computing Infrastructure,” Computational Intelligence and Computing Research (ICCIC), IEEE International Conference.

[12] P. Kumar, & R.K. Pateriya, (2012, July). “A Survey on SQL injection attacks, detection and prevention techniques,” In Computing Communication & Networking Technologies (ICCCNT), 2012 Third International Conference on (pp. 1-5). IEEE.

[13] A. Makanju, A. Zincir-Heywood, & E. Milios, (2012). “A Lightweight Algorithm for Message Type Extraction in System Application Logs,” IEEE Transactions on Knowledge and Data Engineering.

[14] S. Phaltane, A. Nahar, & N. Garge, “Scalable Logging Solutions on Cloud,”

[15] J. Smiy, Shu. Xiaokui , Yao.Danfeng , Heshan, Lin. (2013). “Massive distributed and parallel log analysis for organizational security,” Globecom Workshops (GC Wkshps), IEEE

[16] R. Vaarandi, & M. Pihelgas, (2014, October). “Using Security Logs for Collecting and Reporting Technical Security Metrics,” In Military Communications Conference (MILCOM), 2014 IEEE (pp. 294-299). IEEE.

[17] J.J. Wiley, F.P. Coyle, (2012). “Semantic Hedgehog for Log Analysis,” Internet Technology And Secured Transactions, International Conference IEEE.

[18] I.H.Witten, & E. Frank, (2000). “Weka. Machine Learning Algorithms in Java,” 265-320.

[19] “Apache Lucene Core,” Retrieved April 2, 2015.

[20] AWS “Amazon Elastic Compute Cloud (EC2) - Scalable Cloud Hosting,” Retrieved March 5, 2015.

[21] “Data mining with weka, Part 2: Classification and Clustering,” Retrieved 24 March 2012, from http://www.ibm.com/developerworks/library/os-weka2/

[22] “How to Set Up the ELK Stack- Elasticsearch, Logstash and Kibana,” Retrieved 24 March 2015, from http://knowm.org/how-to-set-up-the-elk-stack-elasticsearch-logstash-and-kibana/

[23] “Logstash grok,” Retrieved 2015, from http://logstash.net/docs/1.0.17/filters/grok

[24] “Powering Data Search, Log Analysis, Analytics Elastic,” Retrieved March 2, 2015.