Spring 2018

# SQL Injection Detection Using Machine Learning Techniques and Multiple Data Sources

Kevin Ross
*San Jose State University*

# SQL Injection Detection Using Machine Learning Techniques and Multiple Data Sources

By: Kevin Ross
Advisor: Dr. Teng Moh
Committee Members: Dr. Melody Moh, Dr. Chris Pollett

CS 298 Report

Department of Computer Science
San Jose State University
San Jose, CA, U.S.A.

Spring 2018

# ABSTRACT

*SQL Injection continues to be one of the most damaging security exploits in terms of personal information exposure as well as monetary loss. Injection attacks are the number one vulnerability in the most recent OWASP Top 10 report, and the number of these attacks continues to increase. Traditional defense strategies often involve static, signature-based IDS (Intrusion Detection System) rules which are mostly effective only against previously observed attacks but not unknown, or zero-day, attacks. Much current research involves the use of machine learning techniques, which are able to detect unknown attacks, but depending on the algorithm can be costly in terms of performance. In addition, most current intrusion detection strategies involve collection of traffic coming into the web application either from a network device or from the web application host, while other strategies collect data from the database server logs. In this project, we are collecting traffic from two points: the web application host, and a Datiphy appliance node located between the webapp host and the associated MySQL database server. In our analysis of these two datasets, and another dataset that is correlated between the two, we have been able to demonstrate that accuracy obtained with the correlated dataset using algorithms such as rule-based and decision tree are nearly the same as those with a neural network algorithm, but with greatly improved performance.*

*Keywords— Network Intrusion Detection, SQL Injection, Machine Learning*

# TABLE OF CONTENTS

# ACRONYMS

**ANN:** Artificial Neural Network

**CFS:** Correlated Feature Set

**CSV:** Comma Separated Values

**HIHAT:** High Interaction Honeypot Analysis Tool

**HTML:** HyperText Markup Language

**HTTP:** HyperText Transfer Protocol

**IDS:** Intrusion Detection System

**kNN:** k-Nearest Neighbors

**KVM:** Kernel-based Virtual Machine

**ML:** Machine Learning

**MLP:** Multi-Layer Perceptron

**OWASP:** Open Web Application Security Project

**PCAP:** Packet Capture

**RAM:** Random Access Memory

**RF:** Random Forest

**SQL:** Structured Query Language

**SSH:** Secure Shell

**SVM:** Support Vector Machine

**TCP:** Transfer Control Protocol

**VM:** Virtual Machine

**WEKA:** Waikato Environment for Knowledge Analysis

# LIST OF FIGURES AND TABLES

# CHAPTER 1. Introduction

Web attacks such as SQL Injection, although they have been around for decades, continue to be a relevant and increasingly damaging cause of exposure of personal data as well as negative financial impact to business and governmental entities [2]. This is true, in particular, as old attacks are modified and evolved, and new attack vectors continue to appear. Industry and security firms devote a great deal of resources to mitigation of web attacks, and many current mitigation strategies have limitations that current research is continually striving to overcome [1].

Much traditional web attack mitigation is done by static analysis of incoming web traffic, also known as signature detection. This strategy involves the creation of a signature characteristic of the web attack and then when this signature is detected, the suspicious traffic can be blocked by a firewall or other security appliance. This method has the benefit of being quick and can be implemented in real time to protect network resources, but one drawback is that only known attacks can be detected.

Another strategy for web attack mitigation specific to SQL injection is to focus on the structure of incoming SQL queries, and if a malformed query is detected, this is considered to be an SQL injection attack. This method has good detection results, and can also detect new attacks that involve malformed queries, but a drawback is that it requires significant knowledge of the application and the structure of what are considered "normal" queries.

An SQL injection detection strategy that is a current topic of research involves the use of machine learning techniques. Popular techniques in this research are decision trees, rule-based learning techniques, support vector machines (SVM), and neural networks. A primary advantage of these techniques is that they are capable of detecting new attacks. A potential drawback with these techniques, however, is the possibility of increased processing time depending on the algorithm used.

Research into the SQL detection techniques mentioned and others rely on the availability of good data. Much current research uses web traffic captured coming in to the web application, or uses logs from the web application and/or web server [3]. The strategy that we are proposing uses traffic captured inbound to the web application in combination with traffic captured between the web application and the associated database server at a Datiphy appliance network node. We are using traffic captured at these two points to create our datasets, and we then create a third dataset by correlating events between the datasets derived from the two capture points.

In the intrusion detection problem space, as well as in any situation involving streaming of large quantities of data, an important concern is the speed and efficiency of operation of a potential classification system. In our project we investigate the use of efficient, quick algorithms to compare their accuracy with those that are potentially more accurate such as neural networks, but with the possible trade off of efficiency, to see if we can combine the speed and efficiency characteristic of signature detection techniques with the potentially increased accuracy and ability to detect new attacks of machine learning techniques.

Another well-known issue with modern IDS research is that realistic attack data can be difficult to gather, and many projects use data from as early as 1999 that is widely considered to be unrepresentative of modern web-based attacks. Many researchers go the route of generating traffic instead of using existing traffic, and many of these projects are using automatic generation tools such as sqlmap. This is a tool that is actually in use by security researchers and attackers, whether directly or as part of a larger project such as Metasploit. In our project, we experimented with generating traffic with sqlmap extensively, but found that certain features tended to be characteristic of this traffic and thus the data proved to be simple to classify. Our approach involves automated generation of realistic traffic incorporating manually coded SQL injection attacks as the basis for machine learning.

The rest of the paper is organized as follows: Section II covers background material and related work. Section III discusses our system design and implementation. Section IV goes into detail about our experiment and results, and Section V concludes our paper.

# CHAPTER 2. Background and Related Work

## *2.1 SQL Injection*

SQL injection is the extension or modification of a web application's SQL statement by an attacker in order to extract or update information in the database that they are not authorized to access [32]. Suppose a webapp generates the following SQL statement:

```
SELECT author,title,year FROM books WHERE publisher = 'Wiley' and published=1
```

If an attacker were to enter a string such as:

```
Wiley' OR 1=1–
```

into the search form, this would result in the following query:

```
SELECT author,title,year FROM books WHERE publisher = 'Wiley' OR 1=1–' and
published=1
```

This query would return every row in the database where the publisher is Wiley, or 1=1, which is always true, thus returning all rows in the database [33].

There is much existing research that attempts to examine and mitigate SQL injection attacks. One technique in use involves static analysis of code to form a model that can be compared to dynamic queries at runtime to detect SQL injection attacks. In Lee et al. [6] the authors propose a system that performs analysis of PHP-based websites to evaluate the SQL queries, then processes these queries by removing parameters to form general query structures, a process they refer to as query transformation. These are then compared to live SQL queries generated dynamically and a query that has a different structure will be flagged as an attack. In addition to removing parameters, Kar et al. [8] generalize SQL queries into structural elements, and were able to achieve a 100% detection rate with their technique.

## *2.2 Architecture*

Many different architectures have been explored for Intrusion Detection Systems (IDS). In Djanali et al. [5], the authors create a clustered architecture using Raspberry Pi computing devices. This project uses the HIHAT honeypot system [9] and the SQL injection detection technique proposed in Lee et al. [8]. The system uses a load-balancing server to route computation to the Raspberry Pi cluster.

Sadasivam et al. [10] propose a distributed, multi-honeypot architecture. They use several different honeypot systems connected to a front-end server, and the system is modular to accommodate the addition or removal of honeypot systems. This distributed architecture has the advantage that it's capable of gathering more data than any one of the honeypot systems, and the authors mention that the attacks they detected were primarily targeting SSH, as well as MySQL, MSSQL, and telnet, and originate primarily in China and the US.

Tawari and Jane [11] propose a complicated architecture they refer to as a "virtual honeynet" designed to maximize the time spent by an attacker in the system and thus the amount of data gathered about attackers. This system uses the SNORT [19] IDS to detect malicious traffic which is then diverted to the honeynet. The virtual honeypots use the HIHAT [9] system as well as the Sebek [30] data capture tool as components, and combine these into a complicated architecture designed to confuse and delay attackers.

In the Honeydoop [12] system, the authors propose a dynamic architecture using Hadoop for data collection. This system uses dynamic allocation based on currently observed network conditions, with the goal of efficient utilization of resources and increased security. The authors mention that in their experience these dynamically created honeypots receive much more traffic than those that are statically generated. The authors use Snort [19] for traffic capture, but the architecture is modular so any IDS system could be used.

## *2.3 Machine Learning*

Machine learning is a technique for automated discovery of patterns in data. In a network security context, machine learning is used to determine the most accurate and efficient way to classify network traffic as an attack or as normal traffic to best protect digital assets while allowing normal business to proceed uninterrupted [31]. There are many machine learning algorithms in use in current research. In Moh et. al [3] the authors are using Naive Bayes and Bayes Net techniques which are creating probability models to classify traffic. In Hanmanthu et al. [13] the authors are using a decision tree technique, and are also evaluating their results in terms of performance characteristics. Another popular technique in current IDS research is the use of Neural Networks, discussed below.

In our project, we're using Decision Tree, Rule-based, Support Vector Machine, Neural Network, and Random Forest algorithms. Background information about each of these is discussed briefly below. We also discuss feature selection, which is the reduction of dimensions in data.

*Decision Tree Algorithms.* Decision tree algorithms are a category of classification algorithms that create a predictive model based on the values of features in the dataset, at each point in the process choosing a feature that divides the dataset to maximize information gain [31], which is a measure of how well each feature is able to predict the class that the data record belongs to, essentially the purity or homogeneity of the covered records in terms of class prediction. A decision tree algorithm iterates through the features of the dataset and chooses the feature with the highest information gain and the best coverage of the dataset, in other words it prefers predictive decisions that contain larger numbers of records. In Hanmanthu et al. [13] the authors are using a custom decision tree technique, and are also evaluating their results in terms of performance characteristics. Their accuracy on SQL injection data is in the 86-87% range.

*Rules-based Algorithms.* Rule-based algorithms are similar to decision tree algorithms in that they are iterating through features of the dataset and selecting those which best predict the class label. The output of rule-based algorithms is a set of rules for classification of the form Condition → Class, an example of which is [31]:

```
(Gives Birth = no) ^ (Aerial Creature = yes) → Birds
```

These rules are evaluated in terms of their coverage and accuracy. Coverage is the ratio of instances of the dataset that are covered by the rules, accuracy is the ratio of the instances that belong to the predicted class to the total instances covered by the rule. Rule-based classifiers are typically comparable in performance to decision tree algorithms, but are considered to be very easy to interpret compared to other classification algorithms [31].

*Support Vector Machine Algorithms.* Support Vector Machine (SVM) algorithms are a classification technique based on statistical methods which work well with high dimensional datasets [31]. SVM attempts to create a decision boundary between classes using instances of training data referred to as support vectors. This is done by finding a maximal hyper-plane separation between instances of the different classes. SVM is considered to be better at finding globally optimal solutions than rule-based or neural network techniques [31]. In Kar et. Al. [7] the authors are using an SVM classifier, and are experimenting with different centrality metrics as input to the SVM classifier. In Choi et. al [16] the authors are using an SVM classifier with n-gram based analysis rather than text-based as we're using for the current project. The authors evaluate SVM in comparison with a kNN (K-nearest neighbor) algorithm and achieve better results with SVM. In Komiya et. al. [17] the authors compare SVM with kNN and also Naïve-Bayes algorithms and find that the best results are obtained with SVM.

*Neural Network Algorithms.* Neural Network algorithms were originally designed to simulate the human brain structure, with a system of linked nodes conceptually related to neurons. At the simplest level, there are input nodes that are exposed to features of the dataset, and output nodes corresponding to the classification outcomes. The output nodes represent a weighted combination of the input, and these weights are updated during the learning process based on learning rate and classification error [31]. A Multi-Layer Perceptron is a type of Neural Network with one or more hidden layers of nodes intermediate to the input and output nodes. An issue with Neural Networks, as mentioned, is the potential slowness of the training phase. "Training an ANN is a time-consuming process, especially when the number of hidden nodes is large." [31]. Another issue discussed in current research is that the results of Neural Networks can be difficult to interpret and in some cases this may leave applications vulnerable to adversaries [15]. In Pinzón et al. [14] the authors are using MultiLayer Perceptron and Support Vector Machine (SVM) techniques in combination to improve accuracy, and compare their results to those of other algorithms such as J48 decision tree and JRip rule-based, discussed below, and have been able to achieve improved accuracy with their technique.

*Ensemble Techniques.* The purpose of ensemble techniques is "improving classification accuracy by aggregating the predictions of multiple classifiers". [31] These techniques use base classifiers such as those mentioned above and then perform classification by using a voting scheme from the predictions made by the base classifiers. One such technique used in this project is random forests, which use decision trees as their base classifier.

*Feature Selection.* Feature selection is used to reduce the dimensionality of data for performance reasons. This is a technique to remove redundant features, which are providing duplicate information of another feature, and irrelevant features, which are providing little or no useful information to help with classification [31]. In Kar et. al., the authors use an information gain technique for feature selection and are using Weka [21]. In our project we're using CFS, correlated feature set, in combination with a genetic search algorithm, both of which are discussed below.

## *2.4 Attack Generation*

Gathering data for SQL injection research is generally done in two primary ways: capturing actual web traffic coming into an organization or honeypot, or the generation of realistic simulated traffic. Both approaches have their advantages and disadvantages. Real web traffic is of course the most realistic, but it can be difficult to determine which packets belong to an attack. It can also be difficult to obtain this type of traffic, as organizations are typically reluctant to share web traffic due to privacy and security concerns. Another issue is that a simple research honeypot may capture mostly automated scans generated by common attack tools that would be more easily captured in a controlled lab setting. Simulated attacks have the advantage that they are controlled so that normal and malicious traffic is easily distinguished for labeling purposes, and a good assortment of attacks can be included based on the latest techniques.

There are many tools in use by researchers in an effort to generate realistic attack traffic to test proposed detection and mitigation strategies. As mentioned, Lee et. al. [8] use the attack simulation tool Paros [29]. The most commonly used attack tool as determined by searches on Google Scholar is SQLMap [28]. Pinzón et al. [14] among others are using SQLMap to generate malicious traffic. Kar and Panigrahi [6] discuss and have tested several examples, including SQLMap and manually coded attacks. Many researchers such as Moh et al. [3], as well as our previous research [4] and the current project, are using manually coded SQL injection attacks to generate attack traffic.

# CHAPTER 3. System Design and Implementation

The approach that we propose in this paper uses machine learning techniques to classify incoming traffic as normal or malicious. The system consists of custom enterprise chat web application with a remote MySQL server backend. Data is captured in two places—HTTP traffic between the traffic generation server and the webapp server is captured, and the resulting MySQL traffic between the webapp server and the remote database server is captured. These two sets of data are then processed and correlated to create a separate dataset containing features from both datasets. Machine learning is done with the Weka Machine Learning Framework, and the machine learning algorithms used are evaluated for classification accuracy as well as efficiency in terms of time to build models and time to classify the training data with 5-fold cross-validation. The general process of our system is illustrated in Fig. 1.

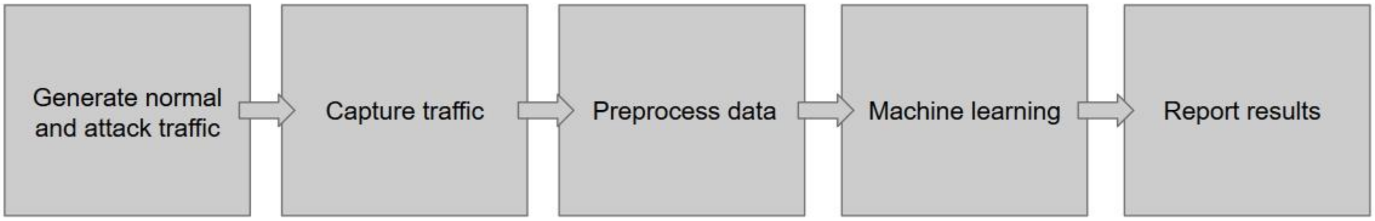

**Figure 1: System Process**

## *3.1 Architecture*

The architecture we're using consists of four server nodes, which are KVM virtual machines running on an HP server with dual quad-core processors and 64G of RAM. These nodes are: a webapp server, a traffic generation server, a database server, and a Datiphy MySQL data capture node. These are illustrated in Fig. 1 and discussed briefly below.

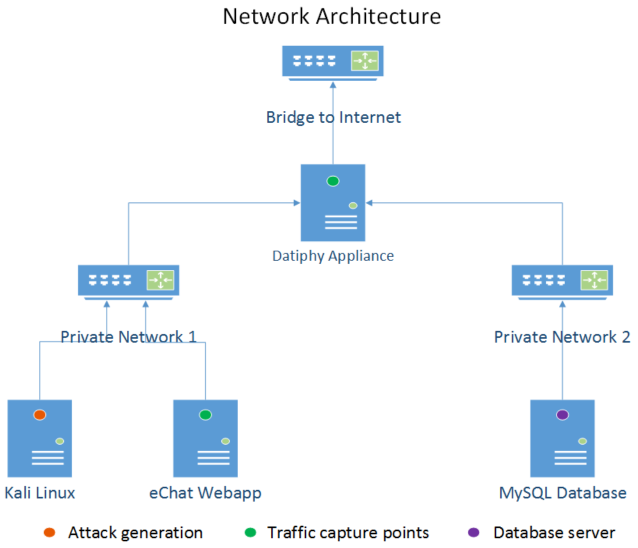

**Figure 2: Network Architecture**

*Web Application Server.* The webapp server is running Ubuntu/Apache, with the custom web application installed in the webspace. This webapp has a MySQL backend which is located on the database server. This server is also running Snort for data capture and is one of our two points of data capture.

*Traffic Generation Server.* This server is used to generate both normal and malicious traffic, and is running the Kali Linux distribution. The normal and malicious traffic is generated with Python/shell scripts using the Beautiful Soup Python libraries.

*Database Server.* This server is running Ubuntu/MySQL. This server is set up for remote access to the database from the chat application on the webapp server, and all MySQL traffic for the webapp occurs between these two servers.

*Datiphy MySQL Data Capture Server.* This server consists of a Datiphy appliance VM provided for research by Datiphy Inc. [18]. This appliance allows for visibility of SQL traffic, among other types of database traffic, and this is how it's being used for this project. Traffic between the webapp and the MySQL database server is routed through this Datiphy appliance, thus allowing for visibility of all traffic in the Datiphy web interface.

## *3.2 Data Preprocessing*

The data generation process we're using consists of three phases: traffic generation, capture, and pre-processing. These are briefly discussed below.

*Traffic Generation.* The simulated normal and malicious traffic for our project is generated from the scripts located on the traffic generation server as discussed previously. This traffic consists of HTTP POST requests from this server to the chat webapp, which then generates MySQL traffic between the webapp server and the database server. Normal traffic consists of simulated normal interaction with the chat web application. Malicious traffic differs in the inclusion of manually coded SQL injection attacks as discussed below. The text is randomly generated to be reasonably realistic in terms of commonality of words and sentence structure in order to simulate statistically realistic web-based communication.

*Traffic Capture.* The traffic is captured at two points, at the webapp server, and at the Datiphy appliance. At the webapp server, we are capturing traffic using the Snort IDS tool which saves its data in the form of PCAP files. At the Datiphy appliance node, the traffic resulting from the interaction between the webapp and the remote MySQL server is captured and a report of this traffic is generated in the Datiphy web interface and saved in CSV format.

*Data Pre-processing.* Data pre-processing is primarily done with bash shell scripts on the webapp server, as this is where the PCAP files resulting from the Snort data capture are located. The PCAP data is processed using TShark, which is a command line interface to Wireshark [20]. TShark allows us to process the PCAP files and extract certain fields of interest from the webapp traffic packets. The result of this is a CSV file, which is then cleaned up with shell scripts for further processing. The data captured at the Datiphy appliance is also processed similarly with shell scripts. In the final stage these two datasets are processed into one file with shell scripts to create the correlated dataset. Correlation is done with the insertion of a token in the form of a unique text string that can be compared and then removed from the dataset once events are correlated.

## *3.3 Web Application*

The web application we're using in our system is a custom coded simulation of an enterprise chat application. This is written in PHP and is running on an Apache webserver. The app uses an HTML form and is configured to use a remote MySQL database backend to store the simulated chat messages. We have designed this webapp to be vulnerable to SQL injection, and the specific vulnerability occurs in an INSERT statement.

## *3.4 Attacks*

The attacks we're using for this project are manually coded SQL injection attacks of several different types, including timing-based attacks, a few types of error-based attacks, and general injection of SQL functions. Timing-based attacks are typically used when there is no immediate display of the results of injection attempts. These attacks, which are also often called blind SQL injection, take the general form:

```
CASE (SUBSTRING(version(),1,1)) when 4 then sleep(5) else sleep(0) ...
```

Successfully injecting code such as this has the effect of introducing a five second delay in the processing of the request when the initial character of the MySQL version is '5'. In our architecture the MySQL version is actually 5.5, so this query will pause five seconds before returning and thus give an attacker information about the database backend of the web application.

An example of one of the error-based attacks we're using is the injection of the following code:

```
... or updatexml (1,concat(0x7e,(user())),0) or ...
```

This code when injected into an SQL statement results in an XPATH error from the database which potentially contains information useful to an attacker, in this case the username under which the webapp is accessing the database. XPath, or XMP Path Language, is a language to interact with XML documents from within MySQL. Common functions used in XPATH SQL injections are UpdateXML() and ExtractValue(), and we're using both for these attacks [33].

Some of the other attacks we're using inject SQL functions such as user(), database(), version(), etc., into the INSERT statement by breaking out of the statement with a single quote character and then injecting the function and proper characters to complete the query, and then inserting characters to comment out the rest of the query. An example of this type of attack is the injection of the following code:

```
',(select database()),now()); --
```

When injected into an INSERT statement, this code will cause the database name to be substituted for the actual input field.

## *3.5 Machine Learning*

Our analysis of the processed data is done with Weka, which is a Machine Learning framework that includes many current ML techniques [21]. Once the data resulting from the pre-processing steps mentioned earlier is imported into Weka, the numerical and nominal data is used as is, and the string data is further processed into word vectors using the Weka filter StringToVec. We then are using Correlated Feature Selection to reduce the number of features to allow for efficient machine learning. This algorithm selects a subset of features that are highly correlated with the class attribute, which in this case is one of our two classes, normal or malicious, but have low correlation with each other. With the CFS algorithm in Weka we're using a genetic search algorithm to find subsets of features to test for the correlation mentioned above. A genetic algorithm is based on the biological concept of natural selection, with a candidate population randomly mutated and tested for fitness, then evolved toward an optimal solution. The genetic search algorithm in Weka is an implementation of the simple genetic algorithm described in Goldberg [22][23].

For the genetic search algorithm, our parameters are as follows: max. generations is set to 200, and population size is also set to 200. In our experiments these values led to the best results. The mutation probability is set to 0.033 and the cross-over probability is set to 0.6—these are the Weka defaults. The number of features for the Snort data is 1611, which is reduced through feature selection to 57. For the Datiphy data, there are originally 1656 features, and this is reduced to 56. For the combined data, the features are reduced from 1662 to 58. The feature reduction led to increased efficiency in our experiments, and was also necessary in our environment to allow the Neural Network algorithm to run in a reasonable amount of time.

For our classification, we are using the J48 decision tree algorithm, the JRip rule-based algorithm, a Random Forest algorithm, an SVM algorithm, and a MultiLayer Perceptron Neural Network, discussed in a previous section. J48 is an open-source implementation in Java of the common C4.5 decision tree algorithm [22]. The decision tree resulting from our experiments for the webapp dataset has 101 nodes and 18 levels, for the Datiphy dataset 97 nodes and 21 levels, and for the correlated dataset 185 nodes and 23 levels. JRip is an implementation of the RIPPER (Repeated Incremental Pruning to Produce Error Reduction) rule-based algorithm proposed by William Cohen [24]. Our ruleset for the webapp dataset consists of 21 rules, for the Datiphy dataset 35 rules, and for the correlated dataset 44 rules. The Random Forests are an ensemble technique which use decision trees as their base classifier. The Random Forest implementation in Weka is based on an implementation by Leo Breiman [25]. The SVM implementation in Weka is from LibSVM [26], which implements a version of the SVM algorithm discussed in Knerr et. al [27]. The MultiLayer Perceptron implementation in Weka is a generic classifier with sigmoid nodes using backpropagation [22]. The Neural Network resulting from our experiments has one hidden layer for each model, with 30 nodes for the webapp dataset, 29 nodes for the Datiphy data, and 30 nodes for the correlated dataset. In our results, presented below, we're evaluating the algorithms for classification accuracy, as well as performance in terms of time spent building the classification model and total time to run on the testing datasets with 5-fold cross validation.

# CHAPTER 4. Experiment and Results

## *4.1 Datasets*

The datasets used in this project consist of simulated normal and malicious traffic originating from the traffic generation server and collected and correlated as previously discussed. For our results the dataset consists of 20000 entries, 10000 normal traffic and 10000 malicious traffic. Unique aspects of each of the three datasets are discussed briefly below.

*Web Application dataset.* This dataset as mentioned was captured inbound to the webapp, and as such some of the unique features include information about the TCP and HTTP packets. Specific features which have been useful for machine learning in our experiments are the tcp.length and http.content_length.

*Datiphy dataset.* The Datiphy dataset is unique as it's capturing features typical of log analysis such as the SQL statement that results from the request to the web application, but it's also capturing and correlating features such as Response Length and Result that are returned from the database server to the web application. To our knowledge this is the first project to use such a dataset.

*Correlated dataset.* This dataset, as a combination of the previous two datasets, combines features from both, and in our experiments the machine learning algorithms are able to use features from both of these datasets.

## *4.2 Results*

Results of our experiments with various machine learning algorithms are summarized in Table 1. Accuracy is the classification accuracy achieved with the various algorithms, Model Time is the processing time to build the machine learning models, and Testing Time is the time to classify the testing dataset with 5-fold cross-validation. Classification accuracy is plotted in Fig. 2, with F-scores presented in Table 2.

### Table 1: Results With 20000 Records

| Dataset | Algorithm | Accuracy | Model Time | Testing Time |
|---|---|---|---|---|
| Webapp | JRip | 94.740% | 3m30.85s | 2.70s |
| | J48 | 95.630% | 1m42.65s | 2.55s |
| | RF | 96.525% | 5m30.20s | 30.60s |
| | SVM | 94.025% | 2m35.80s | 1m10.45s |
| | ANN | 96.715% | 46m03.55s | 3.35s |
| Datiphy | JRip | 95.980% | 6m10.90s | 3.35s |
| | J48 | 96.995% | 1m59.30s | 2.45s |
| | RF | 97.210% | 6m10.20s | 33.50s |
| | SVM | 95.190% | 2m11.50s | 1m3.45s |
| | ANN | 97.285% | 41m23.00s | 2.95s |
| Correlated | JRip | 97.150% | 11m50.80s | 2.10s |
| | J48 | 97.295% | 2m01.80s | 2.05s |
| | RF | 98.055% | 4m22.55s | 35.45s |
| | SVM | 95.715% | 3m30.85s | 1m5.90s |
| | ANN | 97.615% | 47m25.25s | 3.80s |

### Table 2: F-score for 20000 Record Data

| Dataset | Algorithm | True Pos. | False Pos. | True Neg. | False Neg. |
|---|---|---|---|---|---|
| Webapp | JRip | 9117 | 169 | 9831 | 883 |
| | J48 | 9385 | 259 | 9741 | 615 |
| | RF | 9487 | 182 | 9818 | 513 |
| | SVM | 9134 | 329 | 9671 | 866 |
| | ANN | 9501 | 158 | 9842 | 499 |
| Datiphy | JRip | 9291 | 95 | 9905 | 709 |
| | J48 | 9538 | 139 | 9861 | 462 |
| | RF | 9757 | 46 | 9954 | 243 |
| | SVM | 9226 | 188 | 9812 | 774 |
| | ANN | 9613 | 156 | 9844 | 387 |
| Correlated | JRip | 9623 | 193 | 9807 | 377 |
| | J48 | 9656 | 197 | 9803 | 344 |
| | RF | 9860 | 32 | 9986 | 140 |
| | SVM | 9457 | 314 | 9686 | 543 |
| | ANN | 9644 | 121 | 9879 | 356 |

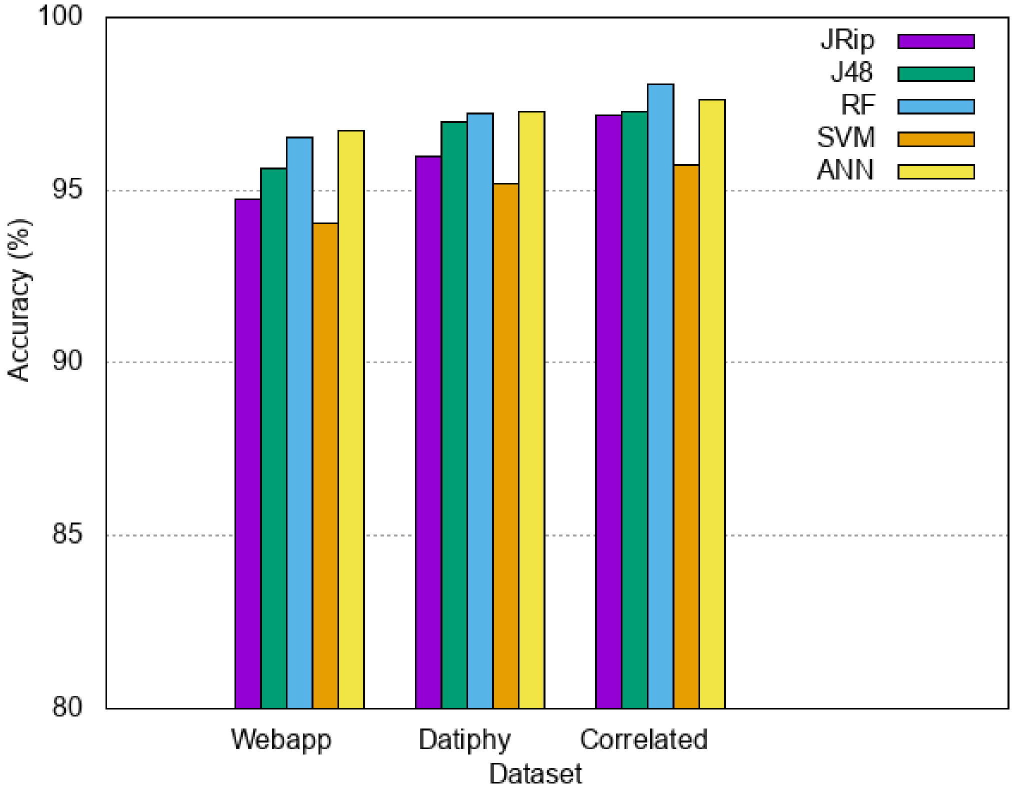

**Figure 3: Classification Accuracy**

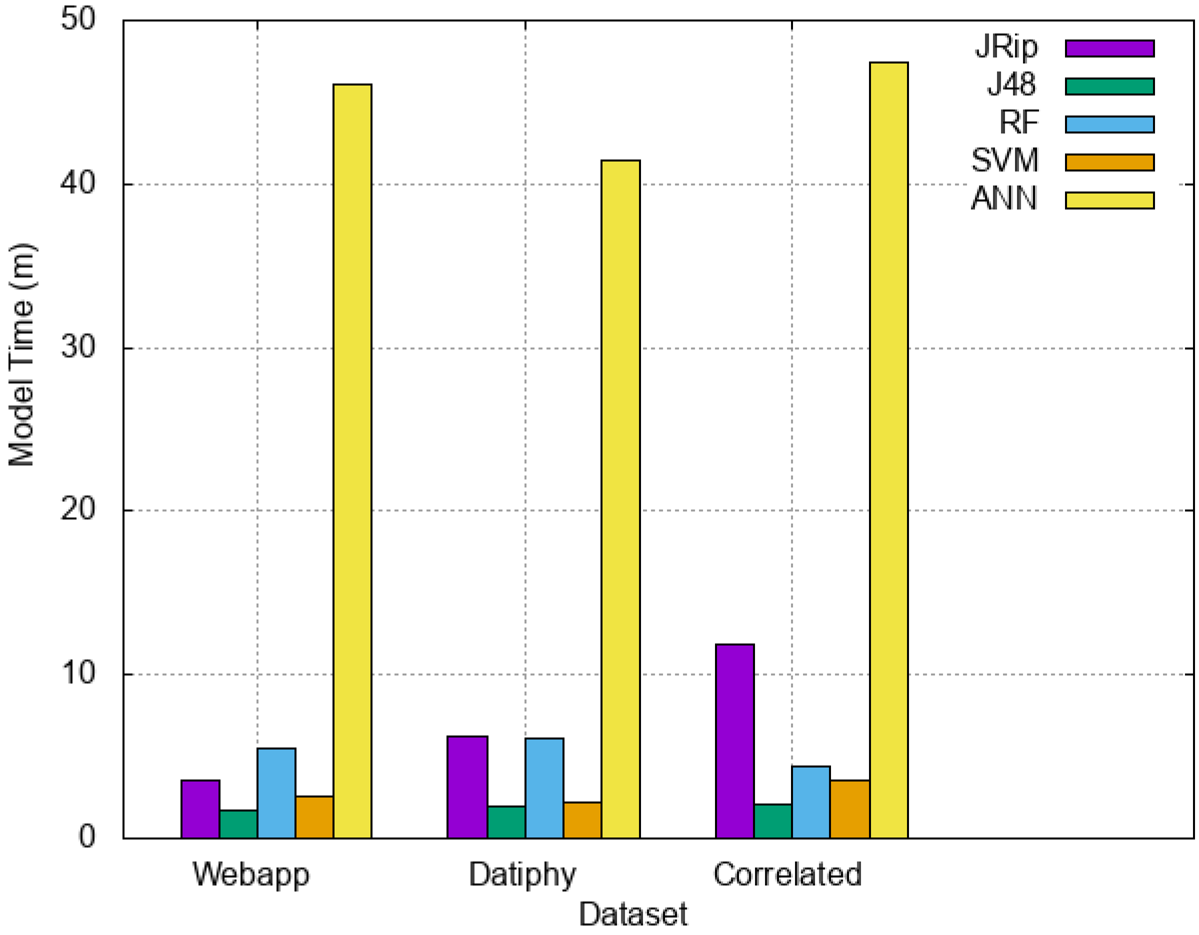

**Figure 4: Modeling Time**

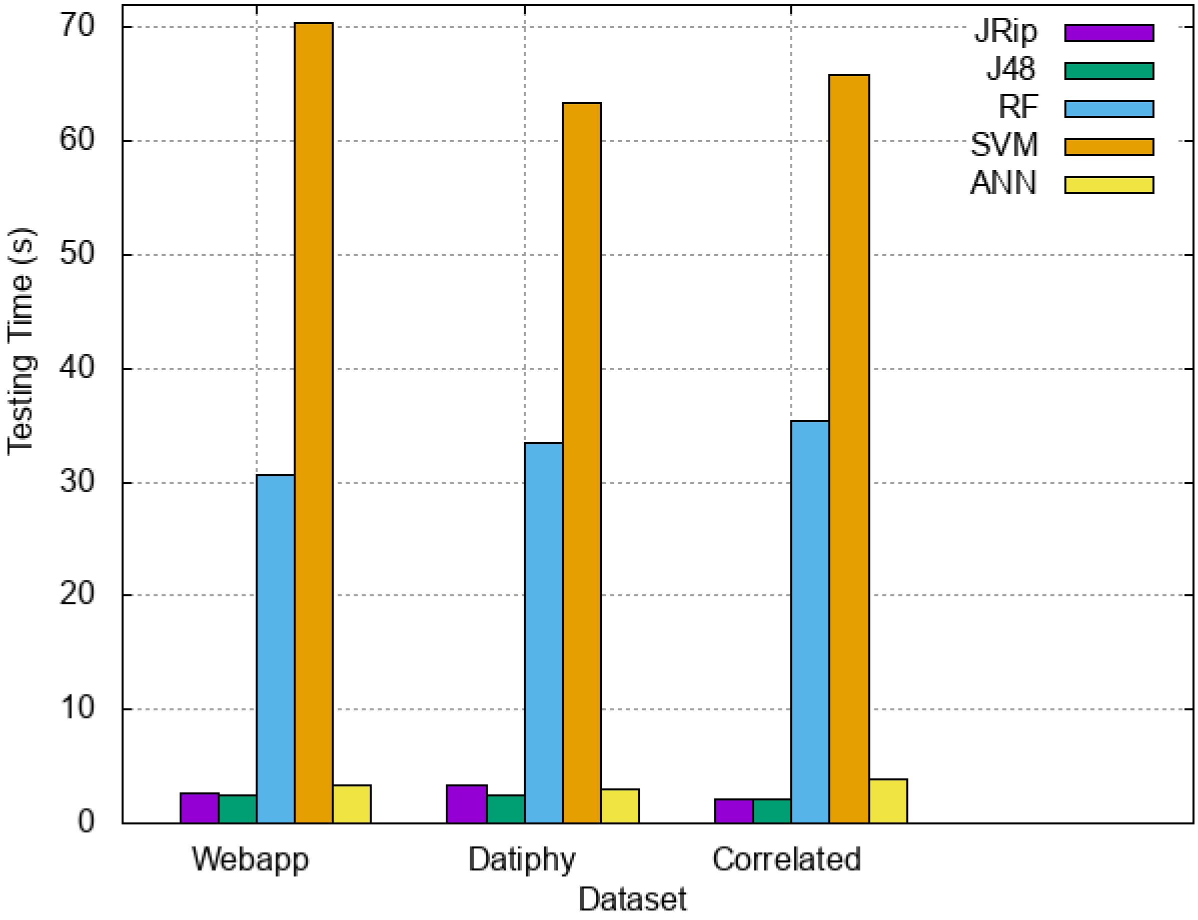

**Figure 5: Testing Time**

# CHAPTER 5. Analysis

An intuitive understanding of this process would suggest that as there's more data available from the correlated dataset, as it's a combination of two datasets, the results obtained would be better in terms of classification accuracy. In our experiments so far this has been the case, with JRip for example using features from both datasets in generating rules as mentioned earlier. As an example, several of the highest ranking rules created from the JRip algorithm are incorporating both the http.content_length from the webapp dataset and the SQL Length from the Datiphy dataset.

In our previous results [4], we used 2000 records for our datasets, and experimented with the JRip and J48 algorithms. Those results were similar in that we're consistently getting much better results with the correlated datasets than with either of the individual datasets alone. We've also seen a consistent improvement in results across all metrics with larger datasets. Although we have randomization in our attack generation, the data is apparently falling into categories that the machine learning algorithms can create a model from, and that capability improves as the algorithms are exposed to more data. For the current project, we have added the SVM, Random Forest, and ANN algorithms, and have also recorded performance data.

One aspect of the performance results to note is the good relative performance of the decision tree algorithm in our experiments. Decision tree algorithms are often considered to be resource intensive, and our thought to explain the results in the current project are that when the data is generally easy to classify based on a few well-selected features, then decision tree algorithms perform quite well relative to other classification algorithms.

Based on our experiments, we've been able to determine that on these datasets the results obtained with the correlated dataset using the faster algorithms are nearly as accurate as our results with the MultiLayer Perceptron algorithm, but with a much lower time of execution. One implication of these experiments is that gathering data from multiple sources can potentially improve classification results as much as or more than the choice of potentially more accurate algorithms.

# CHAPTER 6. Conclusion and Future Work

SQL injection attacks, and web-based attacks in general, continue to be a major issue in the security of financial, health, and other critical data, and this problem only increases in importance as more societal processes become more dependent on the internet. In this project we have proposed a multi-source data analysis system for increased accuracy in detection of SQL injection attacks, and have established that the algorithms we have experimented with such as rule-based and decision tree algorithms have in our experiments achieved accuracy close to that of Neural Networks and are much better in terms of time necessary to build models and execution time when classifying testing data. Future works include collection of additional data such as traffic outbound from the web application to the browser, collection of larger datasets to see if this improves performance, analysis of additional machine learning techniques for both accuracy and performance, and adapting this system to detect other types of web-based attacks.

# References

[1] OWASP Top 10, 2013, "Top 10 2013-A1-Injection", https://www.owasp.org/index.php/Top_10_2013-A1-Injection

[2] Application Defense Center (ADC), 2015, "2015 Web Application Attack Report (WAAR)", http://www.imperva.com/docs/HII_Web_Application_Attack_Report_Ed6.pdf

[3] M. Moh, S. Pininti, S. Doddapaneni, T-S. Moh, Detecting Web Attacks Using Multi-Stage Log Analysis, *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, Bhimavaram, 2016, pp. 733-738.

[4] K. Ross, M. Moh, T-S. Moh, J. Yao, Poster: Multi-Source Data Analysis For SQL Injection Detection, *38th IEEE Symposium on Security and Privacy (IEEE S&P)*, San Jo-se, CA, 2017.

[5] S. Djanali, F.X. Arunanto, B.A. Pratomo, H. Studiawan, S.G. Nugraha, "SQL injection detection and prevention system with raspberry Pi honeypot cluster for trapping attacker," *in Technology Management and Emerging Technologies (ISTMET), 2014 International Symposium on*, vol., no., pp.163-166, 27-29 May 2014

[6] D. Kar, S. Panigrahi, "Prevention of SQL Injection attack using query transformation and hashing," *in Advance Computing Conference (IACC), 2013 IEEE 3rd International*, vol., no., pp.1317-1323, 22-23 Feb. 2013

[7] D. Kar, A.K. Sahoo, K. Agarwal, S. Panigrahi, M. Das, "Learning to detect SQLIA using node centrality with feature selection," *2016 International Conference on Computing, Analytics and Security Trends (CAST)*, Pune, 2016, pp. 18-23.

[8] I. Lee, S. Jeong, S. Yeo, J. Moon, "A novel method for SQL injection attack detection based on removing SQL query attribute values," Mathematical and Computer Modelling, vol. 55, no. 1, 2012, pp. 56-68.

[9] M. Mueter, F. Freiling, T. Holz, J. Matthews, (2008). "A generic toolkit for converting web applications into high-interaction honeypots," University of Mannheim, 280.

[10] G.K. Sadasivam, C. Hota, "Scalable Honeypot Architecture for Identifying Malicious Network Activities," *2015 International Conference on Emerging Information Technology and Engineering Solutions*, Pune, 2015, pp. 27-31.

[11] R. Tiwari, A. Jain, 2012. "Improving network security and design using honeypots." *In Proceedings of the CUBE International Information Technology Conference (CUBE '12)*. ACM, New York, NY, USA, 847-852.

[12] S. Kulkarni, M. Mutalik, P. Kulkarni, T. Gupta, "Honeydoop - a system for on-demand virtual high interaction honeypots," *in Internet Technology and Secured Transactions, 2012 International Conference for*, vol., no., pp.743-747, 10-12 Dec. 2012

[13] B. Hanmanthu, B. R. Ram, P. Niranjan, "SQL Injection Attack prevention based on decision tree classification," *2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO)*, Coimbatore, 2015, pp. 1-5.

[14] C. Pinzón, J. F. De Paz, J. Bajo, Á. Herrero, E. Corchado, "AIIDA-SQL: An Adaptive Intelligent Intrusion Detector Agent for detecting SQL Injection attacks," *2010 10th International Conference on Hybrid Intelligent Systems*, Atlanta, GA, 2010, pp. 73-78.

[15] T, Gu, B. Dolan-Gavitt, S. Garg, "Badnets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain", eprint arXiv:1708.06733, 8/2017, https://arxiv.org/abs/1708.06733

[16] J. Choi, H. Kim, C. Choi, P. Kim, "Efficient Malicious Code Detection Using N-Gram Analysis and SVM," *2011 14th International Conference on Network-Based In-formation Systems*, Tirana, 2011, pp. 618-621.

[17] R. Komiya, I. Paik, M. Hisada, "Classification of malicious web code by machine learning," *2011 3rd International Conference on Awareness Science and Technology (iCAST)*, Dalian, 2011, pp. 406-411.

[18] Datiphy Data Sheet, Dec. 2016, Datiphy Inc., San Jose, CA, The Snort Project, August 28, 2015, http://datiphy.com/resources/data-sheet/

[19] "SNORT® Users Manual 2.9.7.3", https://www.snort.org/documents/snort-users-manual

[20] Angela Orebaugh, Gilbert Ramirez, Jay Beale, Joshua Wright. 2007. Wireshark & Ethereal Network Protocol Analyzer Toolkit. Syngress Publishing.

[21] M. Hall, F. Eibe, G. Holmes, B. Pfahringer, P. Reutmann, I. Witten, The WEKA Data Mining Software: An Update, SIGKKD Explorations, Volume 11, Issue 1, 2009

[22] Weka documentation, Weka.sourceforge.net. (2017). Overview. [online] Available at: http://weka.sourceforge.net/doc.stable/overview-summary.html [Accessed 28 Oct. 2017].

[23] David E. Goldberg, 1989. Genetic Algorithms in Search, Optimization and Machine Learning (1st Ed.). Addison-Wesley Longman Publishing Co., Inc,. Boston, MA, USA.

[24] W. Cohen. 1995. Fast effective rule induction. *In Proceedings of the Twelfth International Conference on International Conference on Machine Learning (ICML'95)*, Armand Prieditis and Stuart J. Russell (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 115-123.

[25] Leo Breiman. 2001. Random Forests. Mach. Learn. 45, 1 (October 2001), 5-32. DOI: https://doi.org/10.1023/A:1010933404324

[26] C. Chang and C. Lin. 2011. LIBSVM: A library for support vector machines. ACM Trans. Intell. Syst. Technol. 2, 3, Article 27 (May 2011), 27 pages. DOI: http://dx.doi.org/10.1145/1961189.1961199

[27] S. Knerr, L. Personnaz, and G. Dreyfus. Single-layer learning revisited: a stepwise procedure for building and training a neural network. In J. Fogelman, editor, Neurocomputing: Algorithms, Architectures and Applications. Springer-Verlag, 1990

[28] sqlmap® Automatic SQL injection and database takeover tool: http://sqlmap.org/

[29] Paros: "A Java based HTTP/HTTPS proxy for assessing web application vulnerability."; http://sourceforge.net/projects/paros/

[30] Sebek data capture tool: https://projects.honeynet.org/sebek/

[31] Tan, Pang-ning, et al., Introduction to Data Mining, Addison Wesley, Pearson Education Inc., 2005

[32] Ed Skoudis and Tom Liston. 2005. Counter Hack Reloaded, Second Edition: A Step-By-Step Guide to Computer Attacks and Effective Defenses (Second ed.). Prentice Hall Press, Upper Saddle River, NJ, USA. 2007

[33] The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws. John Wiley & Sons, Inc., New York, NY, USA, 2007