Improving SQLIA Detection Using Feature Selection and SVM

Ajit Kumar Sahoo, and Khushboo Agarwal
Department of CSE, Silicon Institute of Technology, Bhubaneswar

***Abstract*—Web applications are prone to numerous types of security threats. Among which, SQL injection attack is considered to be one of most prevalent and dangerous. In the previous semester, we worked on an approach to identify SQL injection attacks at the database firewall level, so that it is portable and technology/platform independent. We developed a system that examines run-time queries issued by a web application for execution and allows only benign queries to pass through to the database server using an ensemble of Hidden Markov Model (HMM) - a technique that has been successfully used in pattern recognition domain such as speech and handwriting recognition. In this semester, we intend to apply the concept of Support Vector Machine (SVM), which is extensively used in classification problems in data mining. In our problem we encounter vectors of large dimensions of size around 685, which causes higher performance overhead. If we are able to reduce the dimension of these vectors at the same time keeping the structural integrity of the queries intact, then the processing overhead at runtime may be reduced. We are going to implement feature Selection techniques to obtain a minimal subset of attributes without affecting the accuracy. We will study various feature selection methods available in the literature and experimentally determine the technique that best suits for our problem. The feature selection technique will be applied along with the concept of support vector machine. Our aim is to develop a solution which can be practically and easily implemented in the enterprise scenario without any deployment related difficulties. We shall use PHP, MySQL, LibSVM, and MATLAB etc., and other supporting tools in this project.**

1. Introduction

In this era of internet, we have become completely dependent on web applications like email, e-commerce, social networking, online banking, blogs, forums, online gaming and so on. While the Internet and web applications have made our lives much simpler, security & privacy of the sensitive data in the backend databases has become a big concern. Among various types of security threats to which a web application is exposed to by its nature, SQL injection attack is considered to be most prevalent and dangerous. An attacker can modify the syntax, semantics and behavior of dynamic queries generated by a web application by injecting carefully crafted inputs consisting of delimiters, values and SQL keywords. Through SQL injection, an attacker can obtain unrestricted access to the backend databases and extract the potentially sensitive information contained therein. Nowadays, attackers use sophisticated Botnets which automatically discover vulnerable web pages from search engines and launch mass SQL injection attacks from distributed sources.

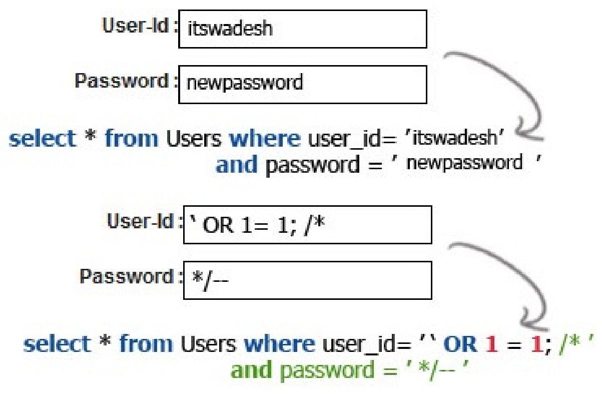
Additionally, recent research by the "Imperva Application Defense Centre" concluded that at least 92% of web applications are susceptible to "malicious attack". The Open Web Application Security Project (OWASP) ranks it on top among the Top-10 security threats. According to TrustWave 2012 Global Security Report, SQL injection was the number one attack method for four consecutive years. About 97% of data breaches across the world occur due to SQL injection alone.

A. SQL Injection

SQL injection [1] is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution. SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. It is mostly known as an attack vector for websites but can be used to attack any type of SQL database.

B. Attack Mechanism

To explain the basic mechanism of SQL injection, we take the classic example of a login form (Fig: 1), commonly seen on many websites.



User-Id : itswadesh
Password : newpassword

```
select * from Users where user_id= 'itswadesh'
        and password = ' newpassword '
```

User-Id : ' OR 1= 1; /*
Password : */--

```
select * from Users where user_id= '' OR 1 = 1; /* '
        and password = ' */-- '
```

Figure 1: SQL injection in a login form

Here the values received from the login form have been directly used to construct the query. If an attacker simply append a tautology to the input, and comment out the remaining, the code is interpreted as:

```
SELECT * FROM Users WHERE username=' ' OR
1 = 1;
```

The comment character at the end of the inputted username, effectively comments out the rest of the query. When executed, the query returns all records from the `users` table as the condition 1 = 1 always evaluates to true.The fundamental fact here is the ability of an attacker to change the syntax and semantics of the query by just inserting special characters and SQL keywords into the input fields, giving him an unauthorized access into the protected area of the website.

C. *Objective of Our Research*

The main objective of the proposed research work is to develop a system that examines run-time queries issued by a web application for execution and allows only benign queries to pass through to the database server.

In our previous work, we have used Hidden Markov Model for detection of SQL injected queries. it was observed that the training time was very high. In this project, we wish to improve the detection process by using SVM as classifer. We wish to reduce the training complexity by reducing the number of features in our dataset.

D. *Organization of the Report*

literature. Chapter 3 states the proposed approach. It gives a brief overview of our system architecture, and all the concepts used in building it. Chapter 4 describes the methodology of our approach. The experimental evaluation of the proposed system is introduced in chapter 5 along with assessment of performance overhead. A comparative study between our previous work and present work is also presented in this chapter. We conclude the report in chapter 6 along with future direction of work.

2. Recent Data Breaches

Despite the fact that numerous methods have been discovered to countermeasure the SQLIA, countless sites are vulnerable to it, and it still causes major breaches. A group of Russian hackers (Known as CyberVor hackers) stole more than one billion passwords from around 400,000 different sites using a botnet in 2014. Some anonymous hackers also hacked the website of the World Trade Organization and leaked personal data of thousands of officials in May 2015. At the same time Indian music streaming service Gaana, which has over 7.5 million monthly visitors, has been compromised by a hacker and its all user information database was exposed. In October 2015, UK's largest ISPs, TalkTalk with around four million UK customers was hit by a severe attack which leads to the leak of customer data including profile information, credit-card details and passwords. UN Climate change summit website also suffered data breach due to this security hole in November 2015. Most recently i.e, in January 2016, nearly 20,000 fans of British electro band Faithless have had their personal details stolen.

Most people are even unaware of that their private information is being compromised everyday due to this simple technique of cyber attack. Not only the organizations suffer huge business losses but also many people make suicide attempts for their private information gets disclosed in public.

3. Literature Survey

Researchers have proposed a wide range of techniques to address the problem of SQL injection. These techniques range from development best practices to fully automated frameworks for detecting and preventing SQL Injection Attacks. In this section, we review these proposed techniques and summarize the advantages and disadvantages associated with each technique.

Various techniques [2] have been proposed for the confrontation of the threat of SQL injection attacks. In this section, the characteristics of the best known techniques are briefly discussed and their principal weaknesses are highlighted.

A. *Defensive Coding*

Proper sanitization of user input data is always advised as the first line of defense against SQLIA. Defensive coding practices consist of techniques applied during application development and require programming in a particular way so that SQL injections cannot happen. Livshits and Erlingsson proposed to use web application construction frameworks and toolkits to protect against code injection attacks, however not all toolkits are secure by themselves. Boyd and Keromytis developed SQLRand which appends a random key to every SQL keyword in the queries used in the application code. The security of this method is limited until the random key is known to the attacker. Gil and Lenz proposed a technique to generate safe SQL queries by using C++ templates. Though the solution is elegant, modern day web applications are developed using other programming languages such as PHP. Nguyen-Tuong et al. suggested automatic hardening of web applications by using precise tainting technique. Taint-based techniques are generally complex to implement from the programmers point of view. Advocating against use of strings for dynamic SQL, Johns et al. presented a language encapsulation technique that provides strict separation between embedded data and code. The approach requires that all legacy, character-based interfaces to the database server be disabled.

Though defensive coding offers first-hand protection against SQLIA, in practice programmers often tend to ignore security aspects due to ignorance or various other limitations during the application development life cycle.

B. *Vulnerability Testing*

Approaches in this category rely on extensively testing web applications to discover security vulnerabilities prior to deployment so that these can be fixed. The earliest work was by Benedict et al., who presented VeriWeb for automatically testing dynamic websites. Kosuga et al. developed SANIA covering syntactic as well as semantic analysis for automatically discovering SQL injection vulnerabilities. Bau et al. conducted a systematic assessment of eight commercial black-box vulnerability scanners . Similarly, Khoury et al. assessed three web vulnerability scanners and presented recommendations to enhance the discovery of persistent SQL injection vulnerabilities. Appelt et al. presented an automated testing approach that applies a set of mutation operators to increase the likelihood of generating successful injection attacks. Vulnerability testing approaches generally require regressive application and their effectiveness is limited by the number of security issues identified.

C. Anomaly Based Techniques

These approaches consist of generating a model of SQL queries and/or the application's behavior during legitimate use within a secured environment. The model is then utilized at run-time to prevent SQL injection by detecting anomalies from the same.

AMNESIA [3] uses a model-based approach to detect illegal queries before their execution into the database. In its static part, the technique uses program analysis to automatically build a model of the legitimate queries that could be generated by the application. In its dynamic part, the technique uses runtime monitoring to inspect the dynamically-generated queries and check them against the statically-built model. The main drawback of AMNESIA is that it requires the modification of the web application's source code. Also it involves a number of steps using different tools.

SQLGuard is based on comparing, at run time, the parse tree of the SQL statement before the inclusion of the user input with that resulting from parse tree of the SQL statement after the inclusion of the user input. A secret key is used for wrapping the user input, so if an attacker compromises this key, SQLGuard is difficult to prevent an attack. Another drawback of their method is that it requires the modification of the application's scripts.

In **SQLRand**, [4] the SQL standard keywords are manipulated by appending a random integer to them. Randomized instances of SQL queries are created by randomizing the template query inside the CGI script and the database parser. To allow for easy retrofitting of existing systems, we introduce a de-randomizing proxy. Code injected by the rogue client evaluates to undefined keywords and expressions. When this is the outcome, then standard keywords lose their significance, and attacks are frustrated before they can even commence.

This technique uses a key to randomize SQL queries, so if an attacker compromises this key, SQLRand is difficult to prevent an attack. Moreover, the approach imposes a significant overhead in terms of infrastructure because it requires the integration of a special proxy in the web-application infrastructure.

All the methods were proved inefficient while dealing with Stored Procedure SQL attacks. Also, Anomaly based approaches generally need access to source code, applicable to a single web application, and heavily dependent on the normal-use model for run-time operation. The normal-use model must be regenerated whenever the application is modified or enhanced, which is a major disadvantage.

4. Proposed Approach

We focus on the link between web application(s) and the database server, i.e., the database firewall layer. Here, the incoming queries can be examined to determine if they are benign or malicious before forwarding to the database server.

Our approach consists of normalizing an incoming SQL query into a sequence of tokens, which is considered as the observation state sequence for input to SVM classifier. The dataset is passed through a feature selection tool to determine the features, which contribute most to the genuinity of a query. Feature selection reduces the dataset dimensinality also to reduce the classifier complexity. The dimensionality reduced dataset is then used to train the SVM model.

A. System Architecture

As any other machine learning method, our system works in two distinct phases: (1) training phase, and (2) run-time phase. The SVMs are first trained using known samples of reduced genuine and injected queries. The model generated is then used at run-time for detecting injection attacks.

1) Training Phase: The training phase (offline) begins with a collection of genuine and injected queries. The process of collecting the queries is described in Section. Fig. 2 shows training of the SVM for queries. The tail-ends of the queries are extracted and normalized. Due to normalization(see section 4.2, several different queries may produce the same sequence of tokens, therefore duplicates are removed. The unique token sequence is then converted into a graph, considering each token as one node of a graph. By calculating the degree of graph, page rank is calculated for each attribute in a query. The dataset preparation process is elaborated in section 5.3. The (attribute, page rank) pair is passed through a feature selection tool to obtain an optimum feature set. SVM is trained by the reduced dataset.(see Section 4.4).

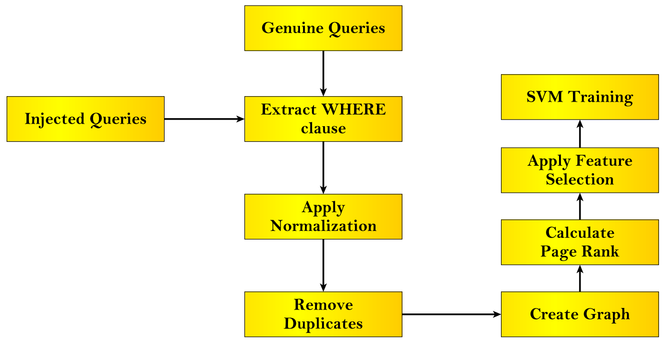


Figure 2: Training of SVM for injected queries

2) Run-time Phase: Fig. 3 shows the architecture of the system at run-time. The tail-end of an incoming query is extracted and normalized into a token sequence. The corresponding numeric index sequence is refined by removing the extra attributes, as per the feature selection technique. The likelihoods evaluated by the SVMs using the prepared model. If the query is determined as injected, it is blocked from execution. On the other hand, if the query is genuine, it is forwarded to database server. The SVM is then retrained using the newly found injected queries, which enables the system to become robust over time.

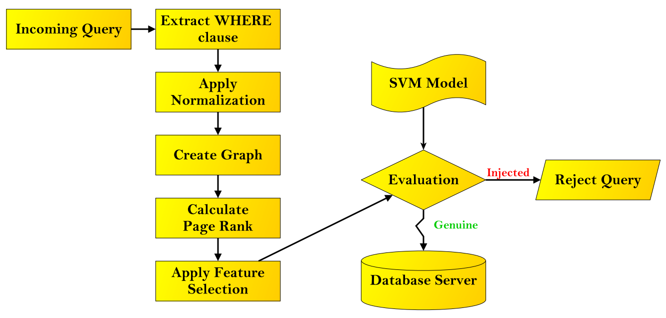


Figure 3: Architecture of the system at run-time

B. *Query Normalization*

Kar et al. proposed a transformation scheme [5] to convert SQL queries into a sentence like form to facilitate textual comparison in a vector space of terms. The scheme normalizes the identifiers, literal values, operators and all other symbols using capital alphabets A-Z and digits 0-9 separating the tokens with spaces.

We incorporate the following enhancements to the transformation scheme considering the following common techniques used by attackers to bypass detection:

1) Replace newline, carriage-return and tab characters with normal space character to neutralize bypassing attempt by white-space spreading.
2) Remove backquotes (`) if any, because these do not contribute towards the structural form of a query.
3) In MySQL, it is syntactically correct to use additional parenthesis-pairs even if not required. For example, `CHAR(65)` and `CHAR(((65)))` both return the character 'A'. To counter this bypass opportunity, matching parenthesis-pairs are removed.
4) Attackers generally embed empty comments to obfuscate injection vectors (e.g, `OR/**/1/**/= **/1`) and bypass detection. However, MySQL allows version specific commands within inline comments. Therefore, we remove empty comments from the query and convert any non-empty comments with their contents into tokens.

After transformation, we additionally perform the following steps to achieve uniformity among queries written using different referencing styles:

1) Substitute `"USRTBL DOT USRCOL"` by `"USRCOL"` to normalize queries written using *TableName.ColumnName* format to the general form.
2) Substitute `"CHR DOT USRCOL"` or `"STR DOT USRCOL"` by `"USRCOL"` to normalize query segments using table aliases (e.g., `P.prodID` or `PR.product_id`)
3) Substitute `"ORDER BY STR"` by `"ORDER BY USRCOL"` to normalize queries where ordering of result is done over an aliased aggregated column.

To visualize the normalization process with these enhancements, consider the following query with a tautological injection vector crafted for bypassing detection:

```
SELECT * FROM `products` WHERE 'prod_id' = 24
oR 'DEF' > CoNcAt(ChAr(0x41), cHaR(0x42),
chAr(0x43));#
```

The normalization process converts it into the following sequence of tokens separated by spaces:

```
SELECT STAR FROM USRTBL WHERE USRCOL EQ INT OR
SQUT STR SQUT GTCONCAT CHAR HEX CMMA CHAR HEX
CMMA CHAR HEX SMCLN HASH
```

The structural composition of a query is thus preserved by the normalization process. For MySQL version 5.5, the complete vocabulary including all keywords, functions, reserved words and the substitutions used in the transformation scheme consists of 685 distinct tokens. We sort the tokens in alphabetic order and sequentially assign an index from 1 to 685 to each token.

Each step of query transformation is a find-and-replace operation, done by appropriately using the preg_replace() and str_replace() built-in functions available in PHP.

Any SQL query, irrespective of its complexity, is thus transformed into a series of words separated by spaces like a sentence in English. The structural form of the query is correctly maintained by the transformation scheme.

C. *Feature Selection*

Feature selection (also known as subset selection) is a process commonly used in machine learning, where a subset of features is selected from the available data for application of a learning algorithm.Selecting the best feature subset is a NP complete problem. The task is challenging because first, the features which do not appear relevant singly may be highly relevant when taken with other features. Second, relevant features may be redundant so that omission of some of them will remove unnecessary complexity. An exhaustive search of all possible subsets of features will guarantee the best feature subset. The best subset contains the least number of features that most contribute towards accuracy.

The two broad categories of feature subset selection have been proposed:

- Filter, and
- Wrapper

Filter techniques assess the relevance of features by looking at the intrinsic properties of the data. In filter criteria, all the features are scored and ranked based on certain statistical criteria. Filter methods are fast and independent of the classifier. They also easily scale to very high-dimensional dataset. As a result feature selection need to be done only once and then different classifiers can be evaluated.

The common disadvantage of filter methods is that they ignore the interaction with the classifier and each feature is considered independently thus ignoring feature dependencies In addition, it is not clear how to determine the threshold point for rankings to select only the required features and exclude noise.

Wrapper methods embed the model hypothesis search within the feature subset search. In this setup, a search procedure in the space of possible feature subsets is defined, and various subsets of features are generated and evaluated. The evaluation of a specific subset of features is obtained by training and testing a specific classification model, rendering this approach tailored to a specific classification algorithm. However, as the space of feature subsets grows exponentially with the number of features, heuristic search methods are used to guide the search for an optimal subset.

Thus feature selection [6] is of considerable importance in classification as it

- Reduces the effects of curse of dimensionality
- Helps in learning the model
- Minimizes cost of computation
- Helps in achieving good accuracy

Information Gain: Information gain [7] is one of the popular approaches employed as a term importance criterion in the text document data. The idea is based on information theory. The information gain of term t is defined in Eq.

$$IG(t) = -\sum_{i=1}^{|C|} P(c_i) log P(c_i) + P(t) \sum_{i=1}^{|C|} P(c_i|t) log P(c_i|t) + P(\bar{t}) \sum_{i=1}^{|C|} P(c_i|\bar{t}) log P(c_i|\bar{t}) \quad (1)$$

where c_i represents the i^{th} category, $P(c_i)$ is the probability of the i^{th} category, $P(t)$ and $P(t)$ are the probabilities that the term t appears or not in the documents, respectively, $P(c_i|t)$ is the conditional probability of the i^{th} category given that term t appeared, and $P(c_i|\bar{t})$ is the conditional probability of the i^{th} category given that term t does not appeared. In this study, before dimension reduction, each term within the text is ranked depending on their importance for the classification in decreasing order using the IG method. Thereby, in the process of text categorization, terms of less importance are ignored, and dimension reduction methods are applied to the terms of highest importance.

D. Support Vector Machine

In machine learning, support vector machines (SVMs, also support vector networks) [8] are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis and outlier detection. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier.

In the classification problem, we attempt to classify points belonging to two given sets in $\Re$ by a linear or nonlinear separating surface. The learning process utilizes the training input data to generate separating surface which assigns each training input data point to the appropriate category. The separating surface is then tested on the unseen data. In order to have a good generalization ability a support vector machine not only minimizes the training error but also attempts to improve the generalization ability of the separating surface.

SVM classifiers starts with a set of data points expressed in cartesian plane.Then, the operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples. Therefore, the optimal separating hyperplane(Refer Fig. 4) maximizes the margin of the training data.

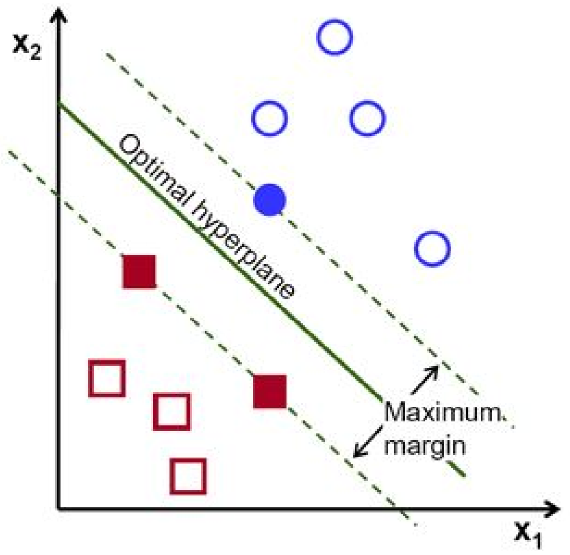


Figure 4: Optimal Hyperplane in SVM

Given l linearly separable training samples $\{\vec{x_i}, y_i\}, i = 1 \ldots l$, where each sample x_i has n attributes ($\vec{x_i} \in \Re$) and a class label $y_i \in \{+1, 1\}$, it finds the optimal hyperplane given by $\vec{w} \cdot \vec{x} + b = 0$ by maximizing the margin between the boundary vectors, so that $y_i(\vec{w} \cdot \vec{x_i}) \geq 1$ for $i = 1 \ldots l$, where $\vec{w}$ is a vector normal to the optimal hyperplane and b is the distance from origin. The linear classifier is given by $f(\vec{x}) = sgn(\vec{w} \cdot \vec{x} + b)$. The optimization problem of the soft-margin linear SVM is given by:

$$\begin{aligned} &minimize && \frac{1}{2}\|\vec{w}\| + C\sum_{i=1}^{l} \xi_i \\ &subjectto && y_i(\vec{w} \cdot \vec{x_i}) \geq 1 - \xi_i \end{aligned} \quad (2)$$

where $\xi_i \geq 0$ are slack variables assigned to each sample and $C > 0$ is a a trade-off parameter between the error and margin. If the data is not linearly separable in the input space, they are mapped into a higher dimensional feature space $\Phi(\vec{x_i})$ where an optimal separating hyperplane can be found. The classifier is then given by:

$$f(\vec{x}) = sgn(\sum_{i=1}^{l} \alpha_i y_i K(\vec{x_i}, \vec{x}) + b) \quad (3)$$

where, α_i are the Lagrangian multipliers, and $K(\vec{x_i}, \vec{x})) = \Phi(\vec{x_i}) \cdot \Phi(\vec{x})$ is known as a kernel function. For linearly separable data, a kernel function can still be used by defining it as $K(\vec{x_i}, \vec{x})) = \vec{x_i} \cdot \vec{x}$. Other kernels, such as polynomial, radial basis function (RBF) and sigmoid etc., are used in classification problems employing SVM. We use the RBF kernel in our approach which is given by:

$$K(\overrightarrow{x_i}, \overrightarrow{x}) = exp(-\frac{|\overrightarrow{x_i}x|^2}{2\sigma^2}) \tag{4}$$

For the SVM solver in our prototype, we use LibSVM(Version 3.21), which is an integrated software library for support vector classification, regression and distribution estimation. LibSVM uses RBF kernel as the default kernel. During training of the SVM, appropriate values for C and γ are supplied.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels

E. WEKA Tool

Waikato Environment for Knowledge Analysis (WEKA) is a popular suite of machine learning software written in Java, used for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. WEKA contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes.

WEKA is open source software issued under the GNU General Public License. WEKA's main user interface is the Explorer, but essentially the same functionality can be accessed through the component-based Knowledge Flow interface and from the command line. There is also the Experimenter, which allows the systematic comparison of the predictive performance of WEKA's machine learning algorithms on a collection of datasets.

5. Implementation

This section gives a detailed overview of our methodology for implemenatation of our approach. We have used PHP and WEKA tool(Version 3.6.13) for our purpose. For SVM classification in our prototype, we use LibSVM(Version 3.21), which is an integrated software library.

A. Query Extraction

Our requirement is a large data set of queries of both injected and genuine type to train and test our model. For the purpose, we developed a web application 'Quiz Portal' and also downloaded several open source web applications built on PHP and MySQL. We run those applications in our local server and extracted the genuine queries from the log file easily. The web applications were also subjected to SQL injection attacks using a mixture of automated scanning and hacking tools such as HP Scrawler (free version), NetSparker (community edition), WebCruiser Pro, SQL Power Injector, sqlmap, The Mole, IronWasp, and jSQL Injector etc. to collect different types of injected queries. We also collected different types of injected queries from various sources in the internet.

After the end of this process, we had 2,53,556 injected queries and genuine queries. Now the problem in front of us was dealing with such a large number of queries. We discovered that most of the queries differ by just some aliases or variable names. So we decided to implement a query transformation scheme.

B. Detection Strategy

The dynamic SQL queries are usually constructed in two parts:

i A static part hard coded by the programmer, and
ii A dynamic part produced by concatenation of SQL keywords, delimiters and received input values.

Since the main objective of a dynamic query is to fetch different set of records depending on the input, the parameter values must be used in the WHERE clause to specify the selection criteria. In fact, SQL injection before the WHERE keyword is extremely rare [9] – not a single instance was found in over 16,500 queries containing SQL injection attacks. Also, data is fetched by SELECT queries in SQL, which implies that unless the injectable parameter is used to construct a dynamic SELECT query that delivers data displayed on the web page, it is not useful for data breach. It points to another assertion that SQL injections mostly occur through dynamically generated SELECT queries.

Therefore, the strategy may be to intercept only SELECT queries for examination; however, this may enable the attacker to cause damage to the data, if not breach. As such, by considering the part of a query after the WHERE keyword, we automatically include UPDATE and DELETE queries in the investigation, because they also support a WHERE clause by SQL syntax.

After this query transformation scheme, we were left with 4609 injected queries and 4884 genuine queries (Massive reduction).

C. Dataset Preparation

In this section, the SQL queries, in the form of string, was converted into a format compatible for data-mining process.

D. Graph Preparation

From the ordered sequence of tokens, a graph $G(V, E, w)$ is generated which captures the structural properties of query as an interaction network of the tokens, facilitating graph analysis using quantitative metrics. The weight of an edge indicates the strength of interaction between two tokens. Since the same token can occur more than once within the sliding window, self loops are allowed. But as per the PageRank algorithm, these are removed. For the edges in a directed graph, we consider left-to-right order as it corresponds to the natural flow of tokens in SQL. As the sliding window proceeds, if an edge already exists, its weight is incremented by the value of the weight function. Size of the sliding window influence the degree to which the interaction between the nodes is captured in the resulting graph. Tokens are considered as related to each other within the window, outside of which the relationship is not taken into account. A smaller window produces a sparse graph, while a larger window produces a dense one. Experimentally we found that the window size of 5 will be best suitable for our problem. Fig. 5 illustrates the tokenization of query using sliding window concept.

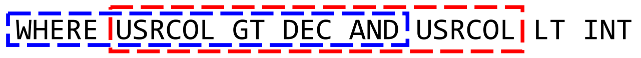


Figure 5: Sliding Window of Size 5

Uniform weighting does not consider the relative distance between two tokens within the sliding window. In proportional weighting, higher weight is assigned to the edge between tokens occurring closer to each other, given by:

$$w_{ij} = s - g - 1 \qquad if\ 0 \langle g \langle s-2 \qquad (5)$$

$$0 \qquad Otherwise \qquad (6)$$

In this case, the weight of the edge between the boundary tokens of the window is 1, and increases by 1 as the gap decreases, so that the weight of the edge between consecutive tokens is the highest. Fig. 6 illustrates the proportional weighting method. The graph of tokens consisting of n nodes

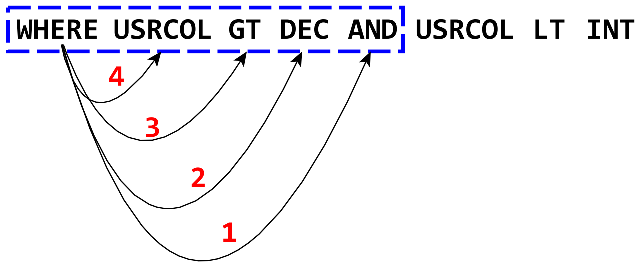


Figure 6: Weight Assignment to Graph

is represented by an adjacency matrix A of size n x n. Rows and columns of the adjacency matrix are indexed by the names of the nodes and each element are $A[t_i, t_j] = w_{ij}$. If the value of $w_{ij} = 0$ then there is no edge between t_i and t_j. Since we consider the directed graph, the rows are the source nodes and columns are target nodes. Fig 7 shows the constructed directed graph of a sample query.

E. Page Rank Calculation

Generally, graph kernels are used in machine learning tasks which involve graphs as samples. Centrality measures such as degree, betweenness, closeness, PageRank etc., have been used in analysis of social networks. We have choosen the most popular PageRank algorithm for our case. The rank

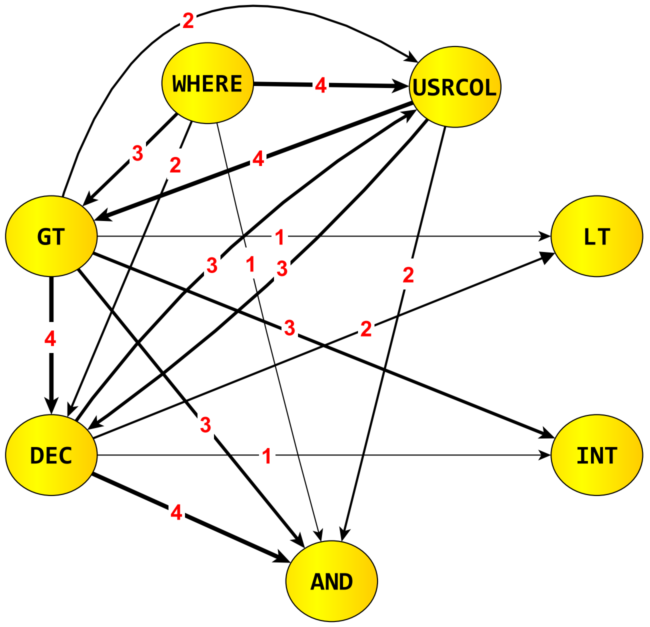


Figure 7: Directed Graph for a Query

of all nodes in a query is calculated using the page rank algorithm and stored in a file. the same is done to all the collected genuine and injected sample queries (9494 numbers) to prepare the dataset for SVM training and testing in SVM light format, where each token of a query is written as (attribute number:pagerank).

F. Information Gain Ranking

For Feature selection, we used WEKA tool by calling *weka.filters.supervised.attribute.AttributeSelection* function:

Algorithm 1: Feature Selection Algorithm

```
Require: Input Train Dataset(80% of genuine and injected
    queries), Test Dataset(Remaining 20%)
1: for ⟨n⟩ = 5 to 150 do
2:     excute command:
       java -Xmx1024m weka.filters.supervised.attribute. At-
       tributeSelection -S "weka.attributeSelection.Ranker -N
       ⟨n⟩" -E "weka.attributeSelection.InfoGainAttributeEval"
       –b -i ⟨Input Train Dataset⟩ -o ⟨Output Train Dateset⟩
       –r ⟨Input Test Dataset⟩ –s ⟨Output Test Dataset⟩ -c 1
3:     Increment ⟨n⟩ by 5
4: end for
```

The options in the above command represent:

-S: ⟨ "Name of search class [search options]" ⟩ Sets search method for subset evaluators.

-E: ⟨ "Name of attribute/subset evaluation class [evaluator options]" ⟩

-N : N top-ranked features

-i : training input file

-o: training output file
-r : test input file
-s: test output file
-c : define which attribute is the class attribute because it is a "nominal" attribute, not a "numerical" attribute and cannot be ranked
-b: batch mode

As we have used *Ranker* as search class for feature selection [10], the required number of class attribute will be moved from top to the end according to their rank.

G. SVM Training & Testing

The *WEKA* feature selection tool gives output as an arff(Attribute Relation File Format) file, which can be directly used for SVM classifier in *WEKA* tool. In WEKA, The default kernel of LibSVM [11] is 'RBF' (Gaussian) kernel, two parameters are important : Gamma and Cost. We train and test SVM model [12], considering all the features and record the result. We also train and test our model by extracting the selected features from the dataset and compare the result to find the optimum feature list for our purpose.

Algorithm 2: SVM Testing and Training Algorithm

```
Require: Reduced Train Dataset(80% of genuine and injected
    queries), Reduced Test Dataset(Remaining 20%) {Obtained
    from Feature Selection algorithm}
1: for ⟨n⟩ = 5 to 150 do
2:     excute command:
       java -Xmx1024m weka.classifiers.functions.LibSVM -d
       "Ouput SVM Model File" -t "Training Dataset with ⟨n⟩
       features" -T "Test Dataset with ⟨n⟩ features" –G 0.8 -C
       32.0 -M 1000.0 -Z -o -i ⟨ "SVM Training and Testing
       Result
3:     Increment ⟨n⟩ by 5
4: end for
```

In the above command, the different options represent:

-C: cost or penalty of training errors.(default 0)

-G: gamma, which control the width of the RBF kernel.(default 1/k)

-Z: Whether to normalize input data, (default) off

-M: Set cache memory size in MB (default: 40)

6. Observation

The model was tested on a laptop computer with $Intel(R)Core-i5^{TM}3230CPU$ 2.60 GHz and 4GB RAM, running Apache 2.2.3 web server with PHP 5.3.28 set up as Apache module, WEKA tool(Version 3.6.13) with LibSVM(Version 3.21) wrapper for classification. The model was setup using a collection of unique 4610 injected queries and 4884 genuine queries. we used 80% from each group for training and remaining 20% for testing the classifier.

A. Pre Feature Selection Results

A SVM model was built by taking unique dataset, containing all of 685 attributes. The results were recorded and shown in table I and table II.

		PREDICTED	
		INJECTED	GENUINE
ACTUAL	INJECTED	TP = 908	FN = 14
	GENUINE	FP = 19	TN = 958

Table I: Confusion Matrix Before Feature Selection

Accuracy	98.21%
True Positive Rate (Recall)	98.50%
False Positive Rate	1.92%
True Negative Rate	98.11%
False Negative Rate	1.50%
Precision	97.92%

Table II: Result Matrix Before Feature Selection

B. Post Feature Selection Results

We applied Information Gain ranking for feature selection, and tested SVM model for different number of features. The graph shows the relationship between the number of features and experimental accuracy obtained.

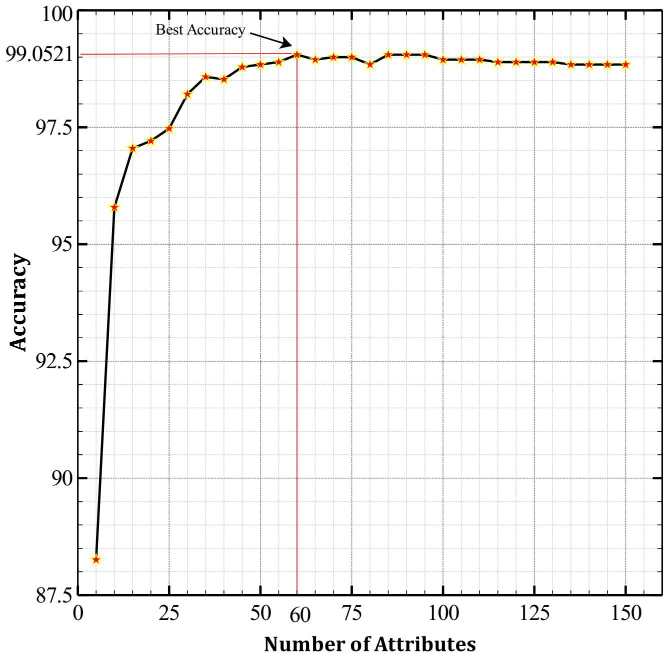


Figure 8: Relationship between #Features and Accuracy

We can see that accuracy is maximum, if we are using 60 features. The detailed result obtained using 60 feature set is shown in table III and table IV.

		PREDICTED	
		INJECTED	GENUINE
ACTUAL	INJECTED	TP = 914	FN = 8
	GENUINE	FP = 10	TN = 967

Table III: Confusion Matrix After Feature Selection

Accuracy	99.05%
True Positive Rate (Recall)	98.98%
False Positive Rate	0.87%
True Negative Rate	99.13%
False Negative Rate	1.02%
Precision	99.18%

Table IV: Result Matrix After Feature Selection

C. Performance Overhead

The time required for various processes were recorded and is shown in Table V and Table VI. wse can see, the overhead is very low, and almost interceptible for end users.

Training Phase (Offline)	
Process	Time/Query(ms)
Query Normalization	2.16
Token Graph & Page Rank	1.15
Information Gain Ranking	0.57
SVM Training	0.21
Total Time required	4.09

Table V: Performance Overhead for Training Phase

Runtime Detection Phase	
Process	Time/Query(ms)
Query Normalization	2.16
Token Graph & Page Rank	1.15
Feature Selection & SVM Testing	0.58
Total Time required	3.89

Table VI: Performance Overhead for Runtime Detection Phase

D. Comparative Analysis

Comparing to our previous SQLIA detection model the overhead reduces to just **3.89ms** from 9.86ms at runtime.The training time also reduced strategically from 2307.65ms to **4.09ms**.

Precision, earlier 98.53% reached 99.18% whereas FPR, one of the biggest concern of our previous model reduced from 5.7% to just 0.87%.

7. Conclusion

This project presented a novel approach to detect SQL injection attacks using a SVM classifier. We have first used query normalization scheme for transforming the queries to simple sentence like form. Then adopted the strategy to examine only the WHERE clause part of run-time queries and ignore INSERT queries. This approach minimizes the size of the legitimate query repository. We represented the queries in the form of graph, based on their interaction with other tokens and then used page rank method to create dataset.By using information gain ranking, we reduced the feature set to an optimum level. The reduced dataset was used to train the SVM. The experimental results are very encouraging and confirm effectiveness of our approach. Performance overhead of the system is almost imperceptible over Internet. The system acts as a database firewall and is able to protect multiple web applications hosted on a shared server, which is an advantage over existing methods. The approach can also be ported to other web application development platforms without requiring major modifications.

In future, we wish to use string kernel [13] along with SVM for detection of SQL injection attacks. String kernel implementation is also present in WEKA tool. False Positives can be reduced to zero. We can use PCA for feature selection and eigen-vector centrality as the metric for SVM training.

References

[1] W. Halfond, J. Viegas, and A. Orso, "A classification of sql-injection attacks and countermeasures," in *Proceedings of the IEEE International Symposium on Secure Software Engineering*, vol. 1. IEEE, 2006, pp. 13–15.

[2] D. A. Kindy and A.-S. K. Pathan, "A survey on sql injection: Vulnerabilities, attacks, and prevention techniques," 2011.

[3] W. G. Halfond and A. Orso, "Amnesia: analysis and monitoring for neutralizing sql-injection attacks," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*. ACM, 2005, pp. 174–183.

[4] S. W. Boyd and A. D. Keromytis, "Sqlrand: Preventing sql injection attacks," in *Applied Cryptography and Network Security*. Springer, 2004, pp. 292–302.

[5] D. Kar and S. Panigrahi, "Prevention of SQL Injection Attack Using Query Transformation and Hashing," in *Proceedings of the 3rd IEEE International Advance Computing Conference (IACC)*. IEEE, 2013, pp. 1317–1323.

[6] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *The Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.

[7] H. Uğuz, "A two-stage feature selection method for text categorization by using information gain, principal component analysis and genetic algorithm," *Knowledge-Based Systems*, vol. 24, no. 7, pp. 1024–1032, 2011.

[8] A. Statnikov, C. F. Aliferis, and D. P. Hardin, *A Gentle Introduction to Support Vector Machines in Biomedicine: Theory and Methods*. world scientific, 2011, vol. 1.

[9] D. Kar, S. Panigrahi, and S. Sundararajan, "Sqlidds: Sql injection detection using query transformation and document similarity," in *Distributed Computing and Internet Technology*. Springer, 2015, pp. 377–390.

[10] G. H. John, R. Kohavi, K. Pfleger *et al.*, "Irrelevant features and the subset selection problem," in *Machine learning: proceedings of the eleventh international conference*, 1994, pp. 121–129.

[11] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[12] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik, "Feature selection for svms," in *NIPS*, vol. 12. Citeseer, 2000, pp. 668–674.

[13] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins, "Text classification using string kernels," *The Journal of Machine Learning Research*, vol. 2, pp. 419–444, 2002.