SQL Injection Attack Classification through the Feature Extraction of SQL Query strings using a Gap-Weighted String Subsequence Kernel

Paul R. McWhirter, Kashif Kifayat, Qi Shi, Bob Askwith

***Abstract*— SQL Injection Attacks are one of the most common methods behind data security breaches. Previous research has attempted to produce viable detection solutions in order to filter SQL Injection Attacks from regular queries. Unfortunately it has proven to be a challenging problem with many solutions suffering from disadvantages such as being unable to process in real time as a preventative solution, a lack of adaptability to differing types of attack and the requirement for access to difficult-to-obtain information about the source application. This paper presents a novel solution of classifying SQL queries purely on the features of the initial query string. A Gap-Weighted String Subsequence Kernel algorithm is implemented to identify subsequences of shared characters between query strings for the output of a similarity metric. Finally a Support Vector Machine is trained on the similarity metrics between known query strings which are then used to classify unknown test queries. By gathering all feature data from the query strings, additional information from the source application is not required. The probabilistic nature of the learned models allows the solution to adapt to new threats whilst in operation. The proposed solution is evaluated using a number of test datasets derived from the Amnesia testbed datasets. The demonstration software achieved 97.07% accuracy for Select type queries and 92.48% accuracy for Insert type queries. This limited success rate is due to unsanitised quotation marks within legitimate inputs confusing the feature extraction. Using a test dataset that denies legitimate queries the use of unsanitised quotation marks, the Select and Insert query accuracy rose.**

***Index Terms*—Intrusion Detection, SQL injection attacks, data mining, String Subsequence Kernel, Support Vector Machine, Supervised Learning**

I. Introduction

SQL Injection Attacks (SQLIAs) involve the crafting of user inputs in order to perform actions beyond the intended function of a web application (Su and Wassermann, 2006). By the identification of the input fields associated with the dynamic generation of queries (Lee et al., 2012; Tajpour et al., 2012), the adversary can probe the database data values, the layout of the database (known as the database Schema), perform remote procedures and escalate their privilege on the Database Management System (Halfond et al., 2006; Balzarotti et al., 2008). Databases often contain significant quantities of confidential information. As a result it can prove to be lucrative for malicious users of web applications to create queries to resolve data they are not authorized to view or alter. SQL Injections are one of the most serious threats to web applications. It is ranked number one in the Open Web Application Security Project (OWASP) Top Ten Application Security Risks in 2013 (Williams and Wichers, 2013). This is due to as many as 98% of web applications having at least one security vulnerability resulting in an increase in SQL injection attacks by ten percent (Trustwave, 2015).

Our solution to the SQLIA problem is the implementation of Machine Learning methods capable of detecting malicious queries based on information from the structure of the query strings learned from a training set of queries produced during runtime. This structural information is extracted using a Gap Weighted String Subsequence Kernel (GWSSK) function (Lodhi et al., 2002). This function computes the similarity of unknown query strings to preselected training query strings. A Support Vector Machine (SVM) classifier uses these similarity measurements to determine if the unknown query is normal or malicious by determining a decision boundary which maximizes the distance between the two classes (Cortes and Vapnik, 1995). Our method is a form of black box method (Halfond et al., 2006).

This method does not require the re-engineering of SQL-dependent web applications or the full disclosure of their source code. This is a flaw of many previous methods (Halfond et al., 2006). There are also some solutions that are easily circumvented by attackers constructing novel attacks (Shahriar et al., 2013). As our method uses a probabilistic classifier in the form of the SVM classifier, unknown queries with query structures which deviate from the training dataset are still likely to be determined as malicious due to the extracted similarity information. Our solution does have two clear limitations. Our method must be placed between the web application and the database. This introduces hardware overhead required to operate the detection and prevention solution (Moosa et al., 2010; Zhang et al., 2011; Pinzón et al., 2013). Additionally, the detection algorithms are never going to have perfect detection accuracy and therefore issues related to false negatives which can inflict database damage and false positives that can prevent normal operation of a database must be mitigated (Makiov et al., 2014).

Our key contribution is the demonstration of the viability of the GWSSK and SVM algorithms in the high-performance classification of SQL query strings during real-time operation

of a database application. This is shown through classification accuracy and time complexity experiments on a dataset of SQL queries exhibiting a wide-range of normal and malicious features. The novel GWSSK method in the automatic extraction of informative features of SQL queries allows for the elimination of biases produced by manually created features potentially improving the accuracy of the SQLIA classification task.

The rest of this paper is structured as follows. In Section 2, the descriptions of related works are presented. In Section 3, the framework of the proposed solution is discussed and the contribution of this paper is clarified. In Section 4, the feature extraction at the core of this solution is defined as the main contribution of this research. In Section 5, the experimental results of the demonstration software for the proposed method are evaluated. These results are then discussed in chapter 6. The final conclusions and proposals of future work are provided within chapter 7.

II. Related Works

Research into securing web applications from SQL Injection Attacks has proposed two differing approaches (Halfond et al., 2006). The first approach involves the rewriting of application source code within the web application and possibly, stored procedures within the database to conduct sufficient input validation. The correct application of these techniques can render a web application secure to injection commands but it comes with a major disadvantage. Completed web applications require redevelopment to incorporate the defensive procedures. However, this is the best way to protect a system from attacks if the system is currently in development and not yet complete. The costs associated with the changing of software vastly increase later into the development cycle.

NoTamper is a black-box testing method designed to determine vulnerabilities in the server-side code. This allows vulnerabilities to be patched although with a severe cost if vulnerabilities are not detected (Bisht et al., 2010a). AMNESIA is another vulnerability exploration method that combines a static analysis of the web application code with runtime monitoring (Halfond and Orso, 2005). SQLGuard was proposed as method of analyzing query parse trees both before and after user-input inclusion. This allows the execution of the user-input to be explored (Buehrer at al., 2005). CANDID is another source code analysis method that retrofits the source code with additional candidate queries. The runtime queries can then be compared to these to determine any illegal executions (Bisht et al., 2010b).

The second approach involves the deployment of additional software designed to screen the queries generated by a web application before their execution on the database. These software solutions utilize a wide range of techniques and are often significantly less expensive to deploy into an active system. Unfortunately, they often suffer from the disadvantage of not being a complete solution to the problem. Many solutions are unable to detect every type of SQL Injection Attack leaving an avenue for attackers to exploit. They can also be prone to false positive and false negative events where the detection algorithms identify legitimate queries as malicious and block them or allowing malicious queries through resulting in a security breach.

SQLProb is a proxy-based architecture to prevent SQL Injection Attacks (Liu et al., 2009). The solution defines a list of queries produced by a web application. It processes all possible queries produced by the typical operation of the web application. These queries are then collected by the proxy software to produce a sample set of SQL queries from the web application. The proxy filter then detects inbound queries and intercepts them. An enhanced Needleman-Wunsch algorithm (Needleman and Wunsch, 1970) originally designed for the alignment of string-based genetic data is used to determine the user input within the full query string. The algorithm determines what substring(s) within the query string to remove to gain the closest comparison to the sample queries. This removed data is the input string(s) within the query string. Upon the determination of the user input, the query string is then used to generate a parse tree. A depth-first-search is then conducted to identify the leaf nodes. If a non-leaf node is discovered that has descendent leaf nodes that are only sourced from the user input then the query string that generated the parse tree is malicious. The malicious queries are then rejected by the proxy software leaving only normal queries to be relayed to the database.

A novel method using the Data-Mining of database logs was proposed to detect SQLIAs (Kim and Lee, 2014). The database log files were used to identify queries executing on the database. This file contains information on the query string and the operations performed by the query execution. The solution first generates a query tree (Buehrer et al., 2005). These query trees were used to generate feature vectors using feature extraction. A set of rules defined by the solution developers transform the string and numerical data from the query tree into a multidimensional numerical vector array. A training dataset of these feature vectors containing samples of normal and malicious queries was used to train a SVM to generate a decision rule for the testing of future queries. Kernel functions were then used to allow the solution to determine a non-linear decision rule. Newly logged queries are transformed into query trees from their associated log, composed into feature vectors and compared by the SVM to the decision rule obtained during the training phase. This solution produced very high accuracy of 99.9% for select and insert queries and 99.6% for stored procedures. The primary disadvantage is that this solution can only be used for attack detection and not prevention. This is due to the simple fact that the query logs that the testing criteria are determined from are only produced when a query is executed.

The combination of static and dynamic analysis techniques were used as the basis of a preventative solution (Lee et al., 2011). In this approach, the source code of a web application is inspected to identify the possible SQL queries. The queries are collected prior to the insertion of user input creating a control query. The solution then dynamically monitors for queries being generated at runtime. These queries are then

processed by an attribute removal algorithm that removes all data from the query that is contained within quotes as these attributes will have no basis on the syntactic form of the query. This reduced query is then compared using an XOR logic operation to the control query gathered during the static analysis. If this operation returns a result indicating that the two queries are different, the user input must have some form of injection input and it is discarded. This approach is accurate and has very low time complexity as the XOR operation is extremely light on processing. Unfortunately it requires a static analysis which must be accomplished by either the analysis of the web application source code or through the use of a proxy server between the user and the web server.

A framework, using a machine learning approach, implements an Intrusion Detection System that learns the patterns of query strings (Valeur et al., 2005). It uses a supervised learning training dataset to produce training models. First the strings are parsed into syntactic trees for feature extraction. Feature vectors are used to produce a model of the parse trees of typical legitimate queries. Then the training set queries are compared to these models and an anomaly score is determined based on how much the training set queries differ from the models. The solution is then able to operate in a detection phase by intercepting new query strings, extracting their features and comparing them to the models to determine the queries anomaly score. If this score is greater than the maximum anomaly score from the training phase, the query is classified as an attack query and logged. The approach proved to be capable of detecting queries that deviated from the normal template due to the injection of commands with a high rate of confidence. This approach is, however, dependent on being supplied with a complete set of legitimate queries during the training phase. Failure to do so will result in false positives as legitimate queries not used for training will have an increased anomaly score. It does mean that the training set need only describe legitimate queries as all those that differ from these queries are rejected as having high anomaly scores. DoubleGuard is an intrusion detection system that implements multitier detection. It models the network behavior between the front-end web application and the back-end database as well as any intermediate servers. This allows the determination of attacks in the event an attacker bypasses segments of the pipeline (Le et al., 2011).

Machine learning solutions have become a popular method for SQL injection attack detection as they allow a probabilistic representation of the problem to be deployed. This strengthens the methods against novel attacks. A neural network solution trained on normal and malicious HTTP requests can be used to classify these requests although the solution required separate instances for each website on shared hosts (Moosa et al., 2010). SQLiGoT represents SQL queries as a collection of token graphs and uses SVMs to detect attacks at the database firewall layer (Kar et al., 2016). This solution does not require multiple instances and is capable of protecting multiple web applications simultaneously. Multiagent systems have been used to produce an intrusion detection system to detect SQL Injection attacks. idMAS-SQL is an architecture that employs a number of algorithms to classify suspicious queries through the use of Machine Learning classifiers including SVM and artificial neural networks (Pinzón et al., 2013).

Our framework also employs machine learning for classifying query requests but through the use of string kernels (Lodhi et al., 2002), we replace the manual engineering of attack features present in other works and instead allow our machine learning system to determine its own solution based on a training set of known queries.

III. SQLIA Detection Framework

Whilst Machine Learning solutions have previously been developed for the classification of SQLIAs, they are all dependent on features carefully designed for the task. This design task, named Feature Engineering, is a powerful method for crafting highly informative mathematical representations of the query data and is almost ubiquitous in Machine Learning tasks. Despite its wide usage, this method can introduce biases into the solution due to the manual intervention such a task requires. Our method replaces this manual design with a novel string kernel approach which automatically converts the input string data into a high-dimensional mathematical form. This form would be impossible to utilize directly and therefore the dimensionality is reduced through the computation of similarity with landmark training strings. The classifier may then use this automated representation to maximize the performance for the given classification task, in this case the detection of SQLIAs. This eliminates any potential bias introduced by human-engineered measurements.

3.1. Design Concepts

The first phase of the operation of the SQLIA detection framework is the collection of SQL statements from the web application. This can be accomplished by routing outbound messages containing the query statements to software utilizing the proposed solution positioned on either the same web server, or an additional proxy server.

These queries are then subjected to a binary classification approach where the class label of the intercepted queries is predicted and actions performed dependent on this prediction, either by rejecting a malicious query or relaying a legitimate query to the back-end database. The prediction is performed by using learning models produced by the identification of discovered patterns within a set of pre-classified training data. In this framework, a Gap-Weighted String Subsequence Kernel function is used to compute the similarity between data. Feature Vectors generated from this similarity computation between each string from the set of training queries are then used to solve the binary classification problem by the identification of patterns in the feature vectors produced by the different classes of query statement. In order to place query statements into a form ready for the string kernel, a data pre-processing phase is performed were the strings are manipulated into forms that emphasize the important SQL features within the strings.

The SVM uses a Kernel Matrix Q_{ij} to perform a training

phase utilizing the training dataset feature vectors to generate a classification model by determining a decision rule that separates the two classes of feature vectors within a multidimensional feature space. Upon the production of this classification model, the SVM is ready to operate in a testing (or detecting) phase. The testing phase is able to use the classification model produced within the training phase to predict a class for an unknown feature vector produced from a new query statement intercepted by the solution.

3.2. Design Architecture

Like many Black Box solutions, this solution requires the introduction of specially crafted input to build up a set of query strings based on the input (Halfond et al., 2006). These query strings are then used as the basis of the production of decision rules to identify legitimate and attack queries. To accomplish this, a set of input features is produced. This set is composed of input strings of both normal and malicious intent. These inputs are then introduced to the web application by identification of the input fields. A string comparison algorithm will identify the total number of queries generated by the web application and link the associated input fields to each query template. The rest of this solution then operates on each individual query and new threads must be activated in order to process the additional queries.

For each individual query, a set of query strings is constructed to determine the morphology of the query within the application code. Each field that was identified to be associated with the query in the previous section is supplied with either normal or malicious input. If any field is supplied with malicious input, the query produced from this input is classified as malicious. Approximately equal numbers of normal and malicious queries must be constructed. The input generator algorithm produces a set of queries based on the inputs from an input features set containing examples of the different forms of injection commands. Each query in this set has been classified based on the input used to generate them as either normal or malicious. This set of queries is the training set as it will be responsible for the creation of the decision rules for the classification of future queries during runtime operation. This process is equivalent to the static analysis from the related solutions but without the requirement of source code access.

The next algorithm is designed to reduce the size of the complete set of possible SQL queries for a web application by manipulating the features of the query strings. This allows multiple similar strings to produce the same 'feature string' which is used for classification. The main difficulty of Black Box methods is describing the completeness of a query system with a sufficient set of allowed queries. This algorithm is capable of reducing the size of the complete set of possible queries. Therefore, the training set can be of smaller size and yet still be an acceptable sample set of the complete query set. It is important that the training set be an accepTable sample set to assist the machine learning algorithms in producing a satisfactory model. During normal operation of a web application the queries generated will contain differing attribute data values in order for users to access the data they require. As a result, this solution uses a modified version of the attribute removal algorithm introduced by a related solution (Lee et al., 2011). This algorithm removes substrings from the input data that have no effect on the syntactic structure of the query string. These must be removed or it is possible that different data values may influence the classification of the string which is unwanted. This algorithm reduces the query string into a form that emphasizes the syntactic features of the query. This algorithm is called the Feature Manipulation Algorithm within the proposed solution.

The reduced query strings from the training set must then undergo a process called feature extraction. This process will convert the queries into mathematical feature vectors that can be used to produce mathematical decision boundaries for the production of training rules. Feature extraction was used in previous research using string to numerical conversion rules (Kim and Lee, 2014). In this solution a much more powerful algorithm is deployed. The Gap-Weighted String Subsequence Kernel Function is a multidimensional algorithm that can compute the similarity between two strings by identifying the occurrence of short sequences of characters of varying scales. It has been shown to be effective for text classification (Lodhi et al., 2002; Homoliak, 2012). This allows the computation of similarity within a feature space of dimensionality $\sum_{k=1}^{p} \Sigma^k$ where Σ is the alphabet of the query strings and p is the maximum length of subsequence used for the evaluation. It is referred to as a String Kernel as it is a kernel function that operates on argument strings instead of vectors already in mathematical form. Each reduced query string within the training set has its similarity value calculated with every query 'feature manipulated string' within the training set including its own string to produce the feature vector for the query. This feature vector is a numerical vector of n dimensionality and each value is the string compared with strings $\{1, \dots, n\}$ from the training set. This represents the large feature space within a n dimension operational space. When this calculation is performed for every query, a total of n feature vectors are generated. These feature vectors can be lined up into rows to produce a $n \times n$ matrix. This is named a kernel matrix and is the input accepted by the SVM during the training phase.

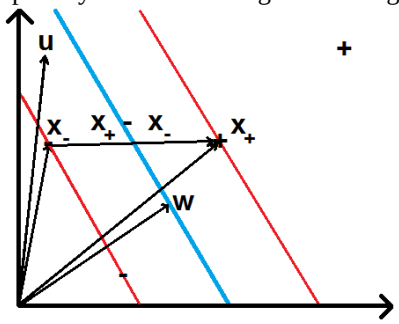


Fig. 1. A two dimensional feature space containing four vectors (two classed negatively, and two positively) and their associated margins.

The solution then makes use of supervised machine learning to utilize the pre-classified training set of feature vectors to generate a decision boundary. The SVM was chosen for this

function as it is a powerful but efficient binary classifier (Cortes and Vapnik, 1995). Consider Figure 1 representing a simple two dimensional feature space containing four feature vectors, two positively classified and two negatively classified.

Within Figure 1 there are a number of important vectors. The vectors $\boldsymbol{x}_-$ and $\boldsymbol{x}_+$ indicate the locations of two support vectors, one classified negatively and one positively. The vector $\boldsymbol{u}$ is an unclassified test vector. Finally, the $\boldsymbol{w}$ vector is a vector normal to the separating hyperplane that describes this hyperplane. The two side lines represent the best fitting margins separating the negative support vectors and the single positive support vector. As the top right hand corner positive vector does not lie on or within the margin of the separating hyperplane, it is not a support vector whereas the other three vectors are.

If the vector $\boldsymbol{u}$ lies upon the positive side of the separating hyperplane, the inner product between $\boldsymbol{w}$ and $\boldsymbol{u}$ is greater than an undefined constant c, $\boldsymbol{w} \cdot \boldsymbol{u} \geq c$. This can be converted to Equation 1 by defining a new constant b where $c = -b$. This equation becomes the first decision rule defined by the hyperplane.

If $\boldsymbol{w} \cdot \boldsymbol{u} + b \geq 0$ then $\boldsymbol{u}$ is a positive classified vector. (1)

This can be expanded for the vectors placed on the margins and outside producing equations 2 and 3.

$\boldsymbol{w} \cdot \boldsymbol{x}_+ + b \geq 1$ where $\boldsymbol{x}_+$ is a positive sample. (2)

$\boldsymbol{w} \cdot \boldsymbol{x}_- + b \leq -1$ where $\boldsymbol{x}_-$ is a negative sample. (3)

An additional variable can be introduced to simplify the Equations 2 and 3 into a single decision rule. Name this new variable y_i such that $y_i = +1$ for positive samples and $y_i = -1$ for negative samples. This produces the new decision rule shown in Equation 4.

$$\boldsymbol{y_i}(\boldsymbol{x_i} \cdot \boldsymbol{w} + b) \geq 1 \quad \text{for both } \boldsymbol{x}_+ \text{ and } \boldsymbol{x}_-.$$
$$\boldsymbol{y_i}(\boldsymbol{x_i} \cdot \boldsymbol{w} + b) - 1 \geq 0 \tag{4}$$

For $\boldsymbol{x_i}$ in the 'gutter', the limit of the margin, Equation 4 is equal to zero. The width of the margins can be defined as shown in Equation 5.

$(\boldsymbol{x}_+ - \boldsymbol{x}_-) \cdot \frac{\boldsymbol{w}}{||w||} \cdot \frac{\boldsymbol{w}}{||w||}$ is the unit vector of $\boldsymbol{w}$. (5)

The margin can be defined independently of the individual vectors resulting in Equation six.

$$\text{WIDTH} = (\boldsymbol{x}_+ - \boldsymbol{x}_-) \cdot \frac{\boldsymbol{w}}{||w||} = \frac{2}{||w||} \tag{6}$$

The best decision boundary will maximize the size of these margins so therefore we must maximize $2/||w||$ which is equivalent to maximizing $1/||w||$ which can then be determined as minimizing $||w||$. For mathematical convenience this is formed into Equation 7.

$$MIN\left[\frac{1}{2}||w||^2\right] \tag{7}$$

This operation can be accomplished through the use of Lagrange Multipliers.

$$L = \frac{1}{2}||w||^2 - \sum \alpha_i[\boldsymbol{y_i}(\boldsymbol{x_i} \cdot \boldsymbol{w} + b) - 1] \tag{8}$$

The derivatives of L must be calculated and set to zero.

$$\frac{\delta L}{\delta w} = \boldsymbol{w} - \sum \alpha_i y_i x_i = 0$$

$\therefore \boldsymbol{w} = \sum \alpha_i y_i x_i$ where α_i gives weighting to the training vector $\boldsymbol{x}_i$. (9)

$$\frac{\delta L}{\delta b} = -\sum \alpha_i y_i = 0 \qquad \therefore \sum \alpha_i y_i = 0 \tag{10}$$

Using Equation 8 and substituting in Equations 9 and 10 results in the production of Equation 11.

$$L = \frac{1}{2}\left(\sum_i \alpha_i y_i \boldsymbol{x}_i\right)\left(\sum_j \alpha_j y_j \boldsymbol{x}_j\right) - \sum_i \alpha_i y_i \boldsymbol{x}_i \cdot \left(\sum_j \alpha_j y_j x_j\right) - \sum \alpha_i y_i b + \sum \alpha_i$$
$$L = \sum \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j$$
Maximize α (11)

This leads to the new decision rule $\sum \alpha_i y_i \boldsymbol{x}_i \cdot \boldsymbol{u} + b \geq 0$ then u is positive.

This Hard Margin Support Vector Machine is very inflexible. It can only create decision rules where the vectors are never allowed to violate the margin boundaries. This can lead to hyperplane overfitting and therefore an overfitting decision rule if any of the support vectors are outliers. A better approach is to use a Soft Margin Support Vector Machine. This approach allows vectors to violate the margins at an associated penalty cost. This can result in a superior decision rule due to better generalization of the models despite the possible incorrect classification of feature vectors in extreme cases. As any vector that manipulates the decision boundary is a support vector, any vectors that violate the margins are also support vectors.

A new cost parameter C is introduced. This parameter identifies the cost associated with the violation of the margin by a support vector $\boldsymbol{x}_i$ by ξ_i. This modifies Equation 7 from the Hard Margin Support Vector Machine into Equation 12.

$$\underset{w,b,\xi}{MIN}\ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi_i$$
Subject to $y_i(\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_i) + b) \geq 1 - \xi_i$ where $\xi_i \geq 0$. (12)

This solution makes use of this Soft Margin Support Vector Machine as it allows the use of the cost parameter to produce better fitting models without overfitting during the training phase. The SVM optimization function is convex meaning that it will not always optimize to the global minima for the model. The SVM implementation used in this proposed solution also

deploys a grid optimization algorithm in order to determine the optimum value of the cost parameter. The model produced by the training set will then be used as the basis of classifying new queries based on the decision rule produced during the testing phase. This is accomplished by assigning either a +1 or -1 to the test queries y_i value. Support Vector Machines are natively linear classifier but as the query feature vectors are likely not linearly separable. Kernel Functions allow feature vectors that are not linearly separable to be separated within higher dimensional space by mapping the feature vectors using a kernel function shown in Equation 13.

Let $\phi(\boldsymbol{x})$ be a transformation of space where $\boldsymbol{x} \in \mathbb{R}^d$, $\phi(\boldsymbol{x}) \in \mathbb{R}^f$ and $f > d$ where f and d are integers.

We want to maximize $\phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j)$ and $\phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{u})$ where $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are the feature vectors of training set points i and j $1 \leq i, j \leq n$ where n is the total number of training set points. Finally, $\boldsymbol{u}$ is the feature vector of a test query.

Propose a Kernel Function:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j) \quad (13)$$

The algorithm is capable of using the decision rules determined from this operation to classify unclassified query string feature vectors based on their position relative to the decision boundary within the feature space. The SVM determines a decision boundary between the normal and malicious query feature vectors such that the margin between both is maximized. This decision boundary is used to create a model that contains the decision rules for future classification. The production of this model signifies the end of the training phase and the solution now operates during web application runtime.

During runtime, real world user input is used to generate queries. These queries are intercepted by the solution and are processed by the Feature Manipulation Algorithm that extracts attribute data that is not of importance to the string syntactic form. It is then processed by the Gap-Weighted Subsequence Kernel Function that generates a feature vector for the new test query string by computing the similarity value of the test query with every query string in the training set. This feature vector is put into a kernel matrix form producing a $1 \times n$ matrix (a row vector created by a transpose of the feature vector). This matrix is then introduced to the SVM running in testing mode. The SVM uses the model generated during the training phase to classify the test query. The query is then logged to file and if the SVM classifies the query as malicious it is rejected. If the query is classified as normal it is then relayed as normal to the back-end database. Figure 26 on the last page of the paper demonstrates the operation of the SQLIA detection framework as well as path of data flow throughout the solution.

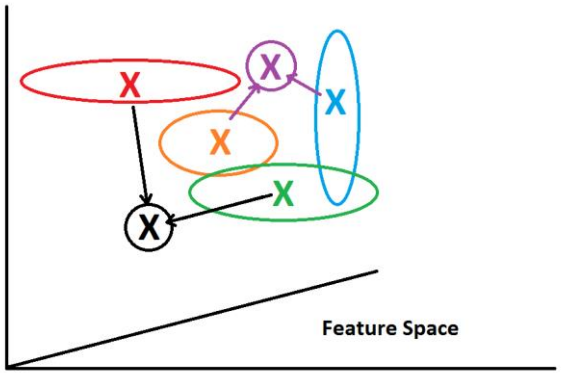
IV. Feature Extraction

SQL queries intercepted by the solution can have a large range of accepTable user input. This user input is of great importance in defining the semantics of a query string but has no effect on the syntactic form of the query. Different types of SQL Injection attacks exhibit the same primary characteristic; the injected input alters the syntactic form of the query. The string kernel function is unable to differentiate between user input used to define attribute values and actual SQL commands isolated from the attributes that alter the function of the query. As a result, the attribute values must be removed from the strings before similarity evaluation. In a previous solution, the attribute values were removed in order to compare the syntactic form of testing query strings to the query template extracted by static analysis of the web application source code (Lee et al., 2011). The Feature Manipulation Algorithm present in this solution is an extension of this original design.

Another reason to remove unneeded substrings from the query string before testing is due to the operation of the Gap-Weighted String Subsequence Kernel function. The total set of characters used between two strings is defined as the alphabet Σ. The time complexity of this function is dependent on this alphabet Σ for the two strings undergoing the comparison. The Feature Manipulation Algorithm can remove the attribute values that are unneeded for the learning process and potentially reduce this alphabet to the reduced alphabet σ where $\sigma \subseteq \Sigma$. This reduced alphabet allows the faster computation of the similarity between the two attribute-removed strings. The string is read in by the function. All double quotation marks are converted to single quotation marks as these SQL operators are interchangeable. This simplifies the next operation, reduces the size of the alphabet and reduces the number of training inputs required to produce a full set of training queries of satisfactory sample size to the complete query set. The algorithm then iterates through the characters. Attribute values are removed by identifying quotations and removing characters after the quotation marks until the next quotation mark is discovered. This prevents substrings that do not affect the syntactic form of the query from being included in the string kernel function. Additionally, numerical data not located within the removed quoted text is identified and converted into the numerical placeholder '1'. This prevents differing numerical data from altering the feature set of the strings. It also simplifies the size of the alphabet and the number of training inputs required. Finally, all characters after the comment operator are removed. This compensates for the ability of comment operators to result in 'uneven sets' of quotation marks disrupting the attribute removal.

These operations performed in string space have a powerful effect on the feature vectors of the query strings. By removing string elements that do not contain syntactic information, the feature vectors of queries demonstrating similar construction are clustered within the feature space allowing for improved operation of the SVM classifier. A similar operation could be accomplished by making use of an unsupervised clustering algorithm on the feature vectors of the query strings and then moving the vectors towards the cluster centroid but at an increased processing requirement. The correct clustering of similar syntactic query strings cannot be guaranteed using unsupervised learning as prior to feature manipulation the feature vectors of similar query strings can be spread over a

large area within the feature space. Figure 2 demonstrates the feature vector clustering effect within the feature space.



Select * from Users where Username = "John" and Password = "JohnsPw";

Select * from Users where Username = "John' or 1 = 1;--" and Password = "";

Select * from Users where Username = "Tom" and Password = "TomsPw";

Select * from Users where Username = "Tom' or 1 = 1;--" and Password = "";

After Feature Manipulation:

Select * from Users where Username = '' and Password = '';

Select * from Users where Username = '' or 1 = 1;#

Fig. 2. A demonstration of the clustering of the feature vectors of similar query strings in the feature space after the processing of the Feature Manipulation Algorithm performing all operations in string space.

The Feature Manipulation Algorithm returns strings with their attribute values removed and with important features enhanced. Feature extraction must be performed on the strings to transform them into numerical feature vectors. Feature extraction uses rules to convert properties of the strings into multidimensional vectors where each dimension relates to a specific property of a string. The SVM requires every output string from the Feature Manipulation Algorithm to be transformed into feature vectors in order to generate models. Given an input query string, new features must be computed depending on the Euclidean distance proximity to 'landmarks - ι' within the input space. Equation 14 demonstrates how the features are constructed.

$$f_n = Similarity(x, \iota_n) \quad \text{where } x \text{ is the input string.} \tag{14}$$

Appropriate 'landmarks' must be chosen to produce a set of features that can appropriately separate the legitimate and malicious manipulated query strings within the feature space, a space of dimensionality equal to the number of features produced by the 'landmark' comparisons. An acceptable method of assigning 'landmark' strings is by selecting each query string within the training set. This is the method utilized in this proposed solution and is the reason why the feature vectors have the same dimensionality as the number of query strings within the training set. Kernel Functions allow classified input vectors that are not linearly separable to be differentiated within higher dimensional space by mapping the inner products between the input vectors using a kernel function. String Kernel Functions are an alternative to explicit feature extraction as they allow the direct computation of the similarity between two strings. String Kernel Functions are defined as the inner products between the features of two argument strings. There are a number of String Kernel Functions that extract specific string features and use them to calculate the similarity value.

The String Subsequence Kernel was published in the Journal of Machine Learning in 2002 (Lodhi et al., 2002; Rouso and Shawe-Taylor, 2005). It was used as part of a novel approach to classifying text documents. These kernel functions use sequence alignment techniques developed for string-based genetic sequence research as an alternative to feature extraction. They consider strings as a collection of symbol sequences. The Subsequence Kernel is based on the identification of a set of sub-sequences within input strings. This allows the calculation of the similarity between two strings by defining a length of substring to identify and producing a multidimensional feature extraction identifying the presence of each possible combination of the alphabet Σ of the string over the maximum subsequence length p and the total dimensionality of the string vectors is given by Equation 15.

$$DIM(GWSSK) = \sum_{k=0}^{p} \Sigma^k \tag{15}$$

The String Subsequence Kernel can be defined through its mapping of k-length substrings between two input strings. The value of this operation will be non-zero if any given string subsequence occurs in both input strings even if it is not contiguous in either of them. All possible characters forming these k-length substrings are collected into an alphabet which is a subset of the complete possible set of characters. Define Σ as a finite alphabet of characters that can be used to construct any string. A string is a sequence of characters from Σ including the empty sequence. For two strings s, t, $|s|$ is the length of string $s = s_1, \dots, s_{|s|}$ and $|t|$ is the length of string $t = t_1, \dots, t_{|t|}$. The string st is defined as the concatenation of the two strings s and t. Further, string $s[i:j]$ is a substring $s_i \dots s_j$ of s.

We therefore can define u as a subsequence of s if there exists indices:
$i = (i_1, \dots, i_{|u|})$ with $1 \leq i_1 < \cdots < i_{|u|} \leq |s|$ such that $u_j = s_{ij}$ for $j = 1, \dots, |u|, u = s[i]$

The length of i in s is $i_{|u|} - i_1 + 1$

Σ^n is the set of all finite strings of length n and Σ^* is the set of all possible strings. This leads to Equation 16.

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n \tag{16}$$

Every possible subsequence of a string can be defined within a feature space of the dimensionality of the alphabet set to the power of the maximum size of subsequence. The dimensions of this feature space is given by $F_n = \mathbb{R}^{\Sigma^n}$.

Feature mapping ϕ for a string s is given by defining the u coordinate $\phi_u(s)$ for each $u \in \Sigma^n$. The value of this

coordinate is given by $\phi_u(s) = \sum_{i:u=s[i]} \lambda^{l(i)}$ for some $\lambda \leq 1$. The variable λ is called the gap decay factor and determines the cost penalty due to non-contiguous substrings. These coordinates measure the number of sub-sequences in the string s weighting them according to their lengths.

The Inner Product of the feature vectors for the string s and t give a sum over all common sub-sequences weighted according to their frequency of occurrence and lengths. This inner product is given by Equation 17.

$$K_n(s,t) = \sum_{u \in \Sigma^n} \langle \phi_u(s) \cdot \phi_u(t) \rangle = \sum_{u \in \Sigma^n} \sum_{i:u=s[i]} \lambda^{l(i)} \sum_{j:u=t[j]} \lambda^{l(j)} = \sum_{u \in \Sigma^n} \sum_{i:u=s[i]} \sum_{j:u=t[j]} \lambda^{l(i)+l(j)} \quad (17)$$

The direct computation involves $O(|\Sigma|^n)$ time and space complexity. This Equation can be used to define a recursive calculation for the kernel shown in Equation 18.

$$K'_i(s,t) = \sum_{u \in \Sigma^i} \sum_{i:u=s[i]} \sum_{j:u=t[j]} \lambda^{|s|+|t|-i_i-j_i+2} \quad \text{where } i = 1, \dots, n-1 \quad (18)$$

Equation 18 is then calculated from 1 to $n-1$ as shown in Equation 19 to 22. Equation 23 then uses this recursive calculation to compute the full subsequence kernel.

$$K'_0(s,t) = 1, for\ all\ s,t, \quad (19)$$

$$K'_i(s,t) = 0, if\ \min(|s|,|t|) < i, \quad (20)$$

$$K_i(s,t) = 0, if\ \min(|s|,|t|) < i, \quad (21)$$

$$K'_i(sx,t) = \lambda K'_i(s,t) + \sum_{j:t_j=x} K'_{i-1}(s,t[1:j-1])\lambda^{|t|-j+2}, \quad i = 1, \dots, n-1, \quad (22)$$

$$K_n(sx,t) = K_n(s,t) + \sum_{j:t_j=x} K'_{n-1}(s,t[1:j-1])\lambda^2. \quad (23)$$

This method penalizes the length of the strings as they grow through the use of the gap decay factor. It is important once this calculation has been performed to normalize the final similarity value. This is important as the length of two strings should be independent of the similarity value. Equation 24 shows how this is performed through the introduction of a new embedding factor.

$$\hat{K}(s,t) = \langle \hat{\phi}(s) \cdot \hat{\phi}(t) \rangle = \left\langle \frac{\phi(s)}{||\phi(s)||} \cdot \frac{\phi(t)}{||\phi(t)||} \right\rangle = \frac{1}{||\phi(s)|| ||\phi(t)||} \langle \phi(s) \cdot \phi(t) \rangle = \frac{K(s,t)}{\sqrt{K(s,s)K(t,t)}} \quad (24)$$

Each string kernel will form different string vectors with different dimensions and for this string kernel we consider a vector with an associated space named 'gap-weighted string subsequence space. Each dimension in this string space is formed by one of the different string combinations determined by Equation 15. Consider a complete alphabet $\Sigma = 2000$, the maximum subsequence length is $p = \Sigma$. This produces a string vector of approximate dimensionality 10^{6602}. However, almost every string will contain a small subset of these substrings resulting in sparse string vectors with most dimensions evaluating to zero. The reduced alphabet, determined by the identification of the alphabet used by the query strings, string vectors will be of significantly reduced dimensionality of approximately 10^{36}. These string vectors will still retain significant sparsity. It is within this space that the Gap-Weighted String Subsequence Kernel will compute the Euclidean distance between the input strings with identical string vectors returning one, dropping to zero as the distance between the string vectors increases towards infinity. This action allows the description of the impossible-to-produce multidimensional vector of string x as a lower dimensional feature vector $\boldsymbol{f}$ shown in Equation 25.

$$\boldsymbol{f} = Similarity(x, \iota_i) \quad \text{where } i \in \{1, \dots, n\}, \boldsymbol{f} \in \mathbb{R}^n \quad (25)$$

Co-occurrences of combinations of the substrings between the two strings result in a higher similarity evaluation. This entire calculation is accomplished without requiring the explicit definition of the two multidimensional string vectors. The Gap-Weighted Subsequence Kernel is similar to the Subsequence Kernel but it also takes gaps between each multidimensional feature into consideration. A gap penalty named the gap decay factor $\lambda \in \{0, \dots, 1\}$ is used to define the reduction in similarity evaluation due to non-contiguity between the co-occurrences of multidimensional features within the two input query strings.

Consider the two strings 'the car parked' and 'at the tree'. The alphabet of these two strings is a set of all the characters within them including the space character. This alphabet is displayed in Equation 26.

$$\Sigma = \{a, c, e, h, k, p, r, t, \_\}$$
Where _ represents the space character. (26)

It is possible to determine the full set of $k = \{1, \dots, n\}$ substrings possible from this alphabet. For $k = 1$ the set is the same as the alphabet. For $k = 2$ the set of possible substrings is shown in Figure 3.

There are Σ^2 possible combinations:

$$\begin{Bmatrix} (a,a)\ (a,c)(a,e)(a,h)(a,k)(a,p)(a,r)(a,t)\ (a,\_) \\ (c,a)\ (c,c)(c,e)(c,h)(c,k)(c,p)(c,r)(c,t)\ (c,\_) \\ (e,a)\ (e,c)(e,e)(e,h)(e,k)(e,p)(e,r)(e,t)\ (e,\_) \\ (h,a)\ (h,c)(h,e)(h,h)(h,k)(h,p)(h,r)(h,t)\ (h,\_) \\ (k,a)\ (k,c)(k,e)(k,h)(k,k)(k,p)(k,r)(k,t)\ (k,\_) \\ (p,a)\ (p,c)(p,e)(p,h)(p,k)(p,p)(p,r)(p,t)\ (p,\_) \\ (r,a)\ (r,c)(r,e)(r,h)(r,k)(r,p)(r,r)(r,t)\ (r,\_) \\ (t,a)\ (t,c)(t,e)(t,h)(t,k)(t,p)(t,r)(t,t)\ (t,\_) \\ (\_,a)\ (\_,c)(\_,e)(\_,h)(\_,k)(\_,p)(\_,r)(\_,t)\ (\_,\_) \end{Bmatrix}$$

Fig. 3. The different possible features of strings utilizing the alphabet of Equation 25.

For a given value of k, the Gap-Weighted Subsequence Kernel can compute the similarity between two strings based on the co-occurrence of k-length substrings by using a dynamic programming approach. This approach also has the advantage of calculating all the similarities for scales between 1 and k without any additional processing overhead. This results in the production of a set of real valued numbers $\{K(1), \dots, K(p)\}$ where $K(k)$ is the computed similarity between two input strings over k-length substrings and p is the maximum length of substrings to be computed. This set of numbers must be used to determine a single similarity value that will be used in the kernel matrix Q_{ij}. These requirements mean that the Gap-Weighted Subsequence Kernel must use a total of $p + 2$ input variables where p is the maximum length of substrings to be used in the similarity evaluation. These variables are the maximum substring length p, the gap decay factor λ which is used to determine how heavily substrings are penalized for not being contiguous within the two input strings and a set of coefficients that determine the weighting of the specific scale similarity evaluations $\{K(1), \dots, K(p)\}$ when they are used as part of a summation to generate the similarity value used for $K(x_i, x_j)$ within Q_{ij}. This normalised summation is shown in Equation 27. This calculation is performed for every x_i and x_j string within the training set to create the Q_{ij} kernel matrix and for a test query u with each training set string x_i to create a 'relative similarity' feature vector for the purpose of the classification of u.

$$K(x_i, x_j) = \frac{1}{\sum_{q=1}^{p} C_{K[q]}} \sum_{q=1}^{p} C_{K[q]} K[q] \qquad (27)$$

Where p is the maximum subsequence size and $C_{K[q]}$ is the weighting coefficient of $K[q]$.

The Kernel Matrix is written into a data file in a format that the SVM library can read demonstrated in Figure 4.

$$\begin{array}{ccccc} y_1 & 0{:}1 & 1{:}K(x_1,x_1) & 2{:}K(x_1,x_2) & j{:}K(x_1,x_j) \\ y_2 & 0{:}2 & 1{:}K(x_2,x_1) & 2{:}K(x_2,x_2) & j{:}K(x_2,x_j) \\ y_i & 0{:}i & 1{:}K(x_i,x_1) & 2{:}K(x_i,x_2) & j{:}K(x_i,x_j) \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$$

Fig. 4. The data file of the training phase Kernel Matrix in the LibSVM format.

It is therefore possible to enter any numerical value to act as the coefficient for the similarity value of a particular subsequence length. This allows solutions to be created that can scale the contribution of subsequence lengths based on their ability to produce a superior classification model. The strength of using a Gap-Weighted String Subsequence Kernel is the ability to compute similarity based on multidimensional features of query strings without the explicit generation of these multidimensional feature vectors. The program simply iterates through the set of possible combinations for an alphabet of all the characters within the two strings up to a given maximum scale length cumulatively summing the contributions as it continues. These multidimensional feature vectors could never be explicitly evaluated for larger scale lengths due to the spatial complexity of such an operation. The memory requirements to store such a large quantity of data would overwhelm any system seeking to make use of this solution. Therefore this string kernel allows the processing of these feature vectors without ever storing them in memory.

V. Experiments

Using the proposed method, a proof-of-concept, fully self-contained C#.NET software capable of generating Select and Insert queries was produced. The software would attempt to classify the generated queries based on models it had created by training on queries generated by passing specially crafted user input through the Select and Insert query generators. At no point was the detection component allowed access to the component containing the unsanitised query template code as this would pollute the objectives of the demonstration software. A SVM capable of utilizing pre-computed kernel inputs was sourced. LibSVM is a library for SVMs and is widely used. This library is equipped with an 'SVC' Support Vector Classification module (Chang and Len, 2007). The LibSVM library is written in Java and therefore a translated library for the .NET languages was required for the C#.NET platform. A library named SVM was utilized. Developed by Matthew Johnson, it is a clean .NET conversion of the LibSVM Java version 2.89. Figure 5 displays a screenshot of the Graphical User Interface of this demonstration software.

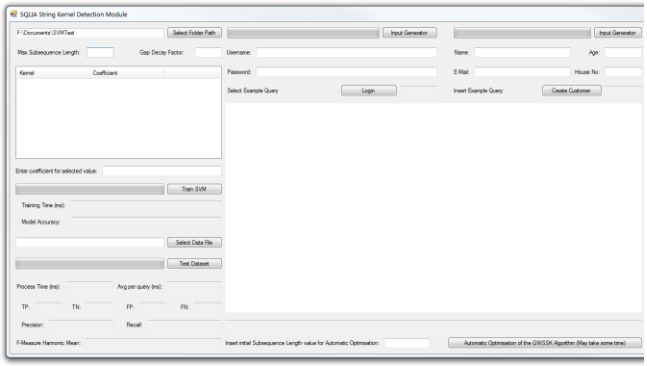


Fig. 5. The Graphical User Interface of the demonstration software.

The evaluation of the proposed solution was conducted on a

machine operating an Intel i7-4770k processor clocked at 4.4 Ghz with 8 Gb of RAM running Windows 7 Professional 64-Bit with Service Pack One installed. As the demonstration software is completely self-contained, no messages are sent over the network and therefore there are no network related time delays. The input feature set data was produced manually and contained values that identified a set of user inputs containing regular input and injection commands combined with a class indicator showing if the input is malicious or legitimate.

5.1. Evaluation conditions

The Amnesia testbed dataset was obtained from the Amnesia authors (Halfond and Orso, 2005). This dataset contains a number of attack queries for seven different web applications. These queries were used to construct two testing datasets, one containing Select queries in the syntactic form of the demonstration software select query and one containing insert queries, again in the syntactic form of the demonstration software insert example query. The Select query dataset contains 232 queries, 116 normal and 116 malicious. These queries feature multiple potential types of SQL Injection Attack and normal queries that attempt to confuse the algorithm by appearing similar to the injection attacks as well as more regular examples. The Insert query dataset follows the same approach but only has 170 queries, 85 normal and 85 malicious, due to a number of types of SQL Injection Attack not being possible without piggy-back type attacks on this form of query string.

Each dataset was tested by computing the peak accuracy, training time and testing time for the $p = 1$ subsequence length which is equivalent to the linear string kernel approach. This gives the ground state accuracy and processing overhead of the Feature Manipulation Algorithm combined with the SVM. The two length subsequence size was then used to generate a full set of detection accuracy and processing time data based on the combinations of possible coefficients weighting the kernel function scale lengths. This was repeated for the three length subsequence size with the length one coefficient locked to one. This set of data was used to determine the effect on the detection accuracy, the rate of false positive and false negative events and the processing time by the different relative weightings of the feature scales. The maximum subsequence size was then increased incrementally by one with the coefficients locked at one to determine the changes to accuracy and processing time by using larger feature scales. These two tests show the relative change in detection accuracy, the rate of false positive and false negative events, the model training time and the query string processing time by using the Gap-Weighted String Subsequence Kernel instead of a simple linear string kernel.

The Evaluation focused on three major indicators of performance. The detection accuracy, given by the occurrence of true positive, true negative, false positive and false negative events used to compute the precision and recall for each testing dataset and finally the F-Measure harmonic mean. The time complexity indicating the amount of processing time required for the evaluation of each query and the spatial complexity identifying the amount of memory required for processing these datasets.

The Precision is the ratio of detected SQL Injection Attacks to the total number of queries classified as SQL Injection Attacks. It is an indication of a bias of the SVM model towards producing false positive results. The more false positive events the model generates the lower the value of the Precision. No false positive events result in a Precision value of one. Equation 28 shows the Precision.

$$Precision = \frac{TP}{TP+FP} \tag{28}$$

The Recall is the ratio of detected SQL Injection Attacks to the total number of actual SQL Injection Attack queries within the testing dataset. It is an indication of a bias of the SVM model towards producing false negative results. The more false negative events the model generates the lower the value of the Recall. No false negative events result in a Recall value of one. Equation 29 shows the Recall.

$$Recall = \frac{TP}{TP+FN} \tag{29}$$

The Precision and Recall together can be then used to generate the F-measure of the testing dataset. This value is a harmonic mean of the Precision and the Recall and is an excellent mechanism for describing the actual detection accuracy of the SVM classification. The F-Measure is given by Equation 30.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{30}$$

The time complexity is an important consideration in the operation of this solution. Using the Stopwatch function, specific regions of code have their operation time recorded. The first stopwatch records the complete processing time per test query and records it into the log file alongside any relevant query information. Further stopwatches were implemented to display processing time information into the program user interface. The complete processing time for the test dataset is displayed alongside the average processing time per query determined by the previous result divided by the number of queries within the testing dataset. A final stopwatch was added to determine the training time for the SVM.

5.2. Detection accuracy

The first test set required the input generator to use the feature dataset to generate a full training dataset using the Select query. The 23 entries within the feature dataset created 86 Select queries of which 46 were legitimate queries and 40 were malicious SQL Injection Attacks. These queries were then used for training a model for the ground-state $p = 1$ case. This reflects the operation of the Feature Manipulation Algorithm and the SVM operating with a linear string kernel. This linear string kernel counts the co-occurrence of characters within the two strings undergoing comparison. This

trained model took 2037 milliseconds to train. The Amnesia dataset derived Select query dataset was then classified using this model. There were 232 total queries of which 116 were normal queries and 116 were malicious queries. 114 of the malicious queries were successfully identified leaving 2 false negatives. The classification of the normal queries was less successful with only 57 correctly classified leaving 59 false positives. This gave the ground state a precision of 65.9% and a recall of 98.3%. The F-measure detection accuracy was 78.9%.

The same test was then performed for the Insert query. As the Insert query within the demonstration software featured four user input locations, the 23 feature dataset entries created 132 training set queries of which 80 were normal queries and 52 were SQL Injection Attacks. A new model was trained using this dataset and took 6563 milliseconds to train. The Amnesia testbed derived Insert query dataset was then classified using this model. There were 170 total queries of which 85 were normal queries and 85 were malicious queries. All 85 malicious queries were successfully classified leaving zero false negatives. However, only 45 normal queries were correctly classified leaving 40 false positives. This gave the ground state a precision of 68% and a recall of 100%. The F1 detection accuracy was 81%.

The difficulties detecting the normal queries were due to unsanitised quotation marks within the normal queries. These queries contained SQL code but not in a position where they would produce injection commands when concatenated into the query strings. However, the presence of quotation marks in the string still caused overly detailed feature manipulated strings to be introduced to the string kernel algorithm resulting in confusion. As most regular user input into query strings does not use quotation characters, a second set of Select and Insert test queries were produced that mirror the first testing set but the normal queries lack quotation mark input. The $p = 1$ test used above was then applied to these two new datasets named Select-Fix and Insert-Fix. The Select-Fix dataset when tested on the previously trained model resulted in a ground state detection accuracy of 99.1%. Using the previous model on the Insert-Fix dataset resulted in a ground state accuracy of 100%.

Next all four test datasets (Select, Insert, Select-Fix, Insert-Fix) were used to generate surfaces for the $p = 2$ state. In this state there are three additional variables, the gap decay factor and the coefficients for length one and length two features. The gap decay factor can take values between zero and one. It was found that this variable made very little difference to detection accuracy so long as it was kept under 0.5. The detection accuracy begins to drop to the linear string kernel state if the gap decay factor is set higher than this value. Therefore the value of the gap decay factor was set to 0.0001 and remains so for the rest of the evaluation.

The coefficients for the scaled features can be of any value but as it is the proportionality between the coefficients that determines the relative weighting of features, values between plus one and minus one with a gap of 0.2 were used to generate 121 possible combinations. The SVM was retrained generating 121 different models for these possible combinations and the four datasets were applied to these models to determine their detection accuracy for the $p = 2$ state.

The Select dataset resulted in a peak accuracy of 98.3% with the coefficient of scale one features at 0.4 and scale two features at 1.0. The surface plot of this evaluation showing the change in detection accuracy against the range of possible coefficient values is shown in Figure 6. The plot clearly shows that the strongest peaks of accuracy occur as the coefficients are similar in value and the largest troughs occur when they are opposite in value. This is due to the constructive interaction of both scale sizes when summing to produce the similarity evaluation. The model is able to train on features of both sizes as they both contribute strongly to the similarity evaluation. When both coefficients are opposite in value the features neutralize leaving the SVM with very little useful information to train on resulting in a heavy loss of accuracy. The accuracy seems to vary diagonally across the plot from peaks to troughs and to peaks again. This is because it is not the value of the coefficients that are important but only their relative proportionality. For the Select dataset, adding two-length sub-sequences to the similarity evaluation results in a substantial jump in detection accuracy.

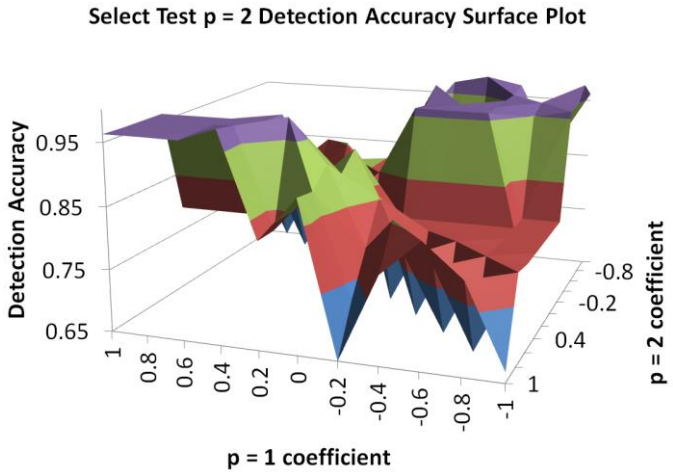


Fig. 6. The p = 2 state detection accuracy surface plots of the Select test dataset.

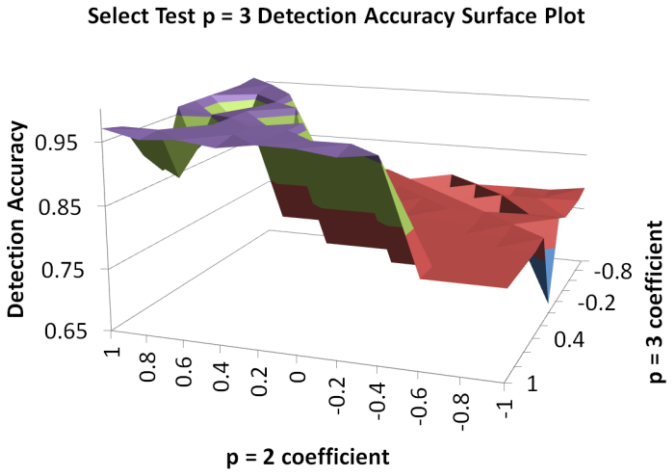


Fig. 7. The $p = 3$ state detection accuracy surface plots of the Select test dataset.

The analysis was then extended to the $p = 3$ state. As positive values for the coefficient of one length sub-sequences proved to provide a boost in detection accuracy for the $p = 2$ evaluation, this coefficient was locked to one and the coefficients for the two and three length sub-sequences were varied next. This did not contribute to the peak accuracy of the Select dataset compared with the previous result of 98.3%. However, this set of coefficients was limited by the locked coefficient for the one length sub-sequences and a high accuracy was still maintained. Figure 7 demonstrates this $p = 3$ experiment and the same shape of surface can be seen as the positive values of the two and three length feature coefficients result in a peak over the ground state accuracy and the negative values result in decay to the ground state accuracy as the higher dimensional features similarity evaluations cancel each other out. These experiments prove that there exist solutions to the gap-weighted subsequence kernel that enhance the accuracy of the Select queries over the linear string kernel.

This analysis was then performed on the Select-Fix dataset. As the linear string kernel accuracy was much higher on this dataset, the multidimensional feature extraction did not produce quite so obvious a set of peaks. However, there existed multiple solutions to the gap-weighted subsequence kernel in the $p = 2$ state that resulted in an accuracy of 100% compared to the ground states 99.1%. The $p = 3$ state was a similar shape also showing the presence of 100% detection accuracy solutions. The surface plots of these two analyses can be seen in Figures 8 and 9.

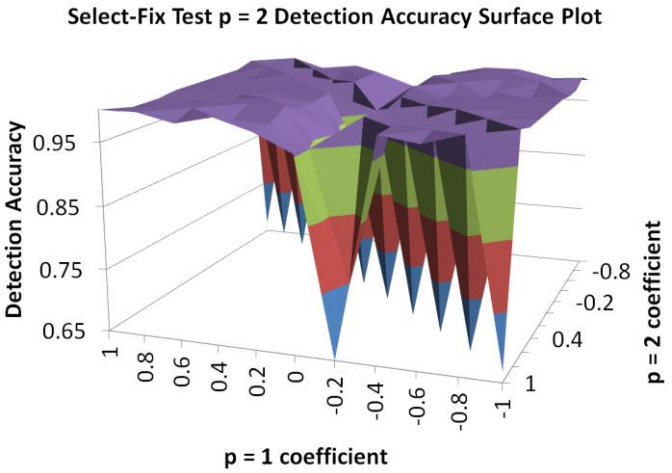


Fig. 8. The p = 2 state detection accuracy of the Select-Fix test dataset.

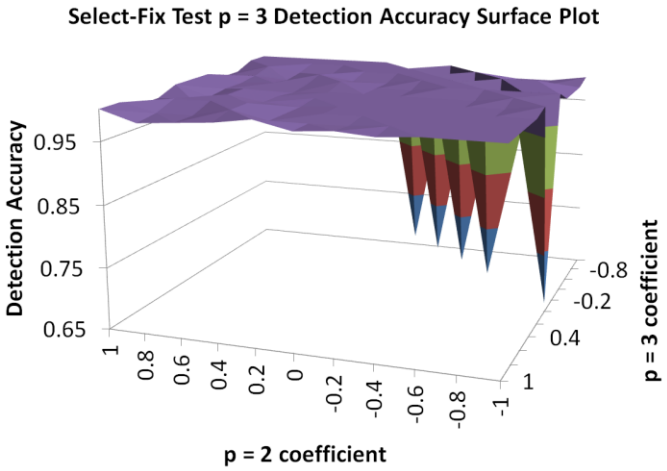


Fig. 9. The p = 3 state detection accuracy of the Select-Fix test dataset.

Next, the Insert dataset was subjected to the $p = 2$ test. The Insert dataset resulted in a peak accuracy of 88.4% with the coefficient of scale one features at minus 0.2 and scale two features at minus 0.8. The surface plot of this evaluation showing the change in detection accuracy against the range of possible coefficient values is shown in Figure 10. The Insert queries did not benefit from the multidimensional feature extraction to the same degree as the Select queries did. However, solutions existed that improved over the linear string kernel detection accuracy.

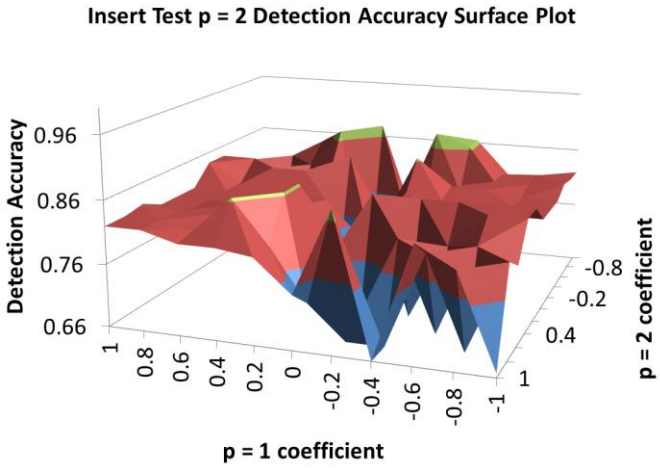


Fig. 10. The p = 2 state detection accuracy of the Insert test dataset.

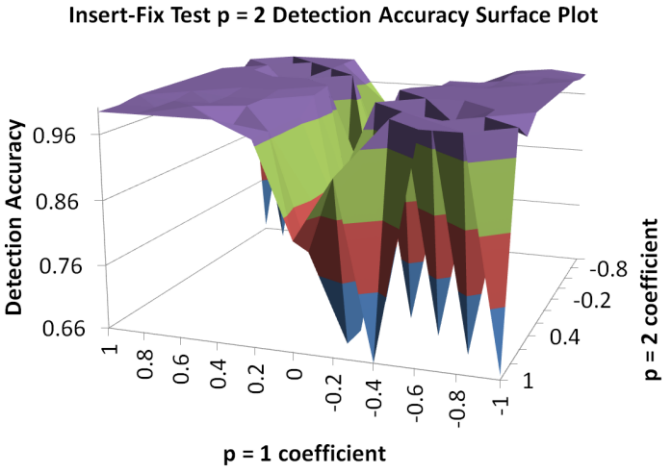


Fig. 11. The p = 2 state detection accuracy of the Insert-Fix test dataset.

As the Insert-Fix dataset had already achieved an accuracy of 100% in the ground state linear state model, using higher dimensional feature extraction was not going to improve the model. In fact, as can be seen in Figure 11, the $p = 2$ state resulted in multiple solutions that have a loss of accuracy from the linear ground state. This demonstrates that the proposed solution in its current form is best for simple Insert-Fix queries when using a linear string kernel. As for the original Insert dataset without quotation marks restricted, the higher dimensional feature extraction does produce solutions with higher detection accuracy. However, as discussed above, superior accuracy may not be the best solution if the rate of false negatives increases. Unfortunately for the solutions with 88.4% accuracy and others with 86.4% accuracy have resulted in false negative events compared with the ground state with zero events. The result data does however show solutions with a mild boost in accuracy over the ground state from 81% to 84.1% whilst still maintaining zero false negative events. Therefore, despite the confusion caused by the poorer multidimensional feature extraction, higher dimensional solutions do exist that improve upon the ground state albeit at a much less impressive level as the solutions for the Select type queries. The strength of the multidimensional feature extraction is dependent on the accuracy of the linear string kernel. Poorer models using this kernel allow for more improvement when using higher dimensional features.

The previous experiment shows that there are solutions for the $p = 2$ and $p = 3$ states that improve on the ground state detection accuracy without introducing new false negative events. The Select type query classification is greatly strengthened by the higher dimensional feature extraction. The Insert type queries do not gain as much of an accuracy increase but still reduced the number of false positives by a small amount. Meanwhile the datasets that removed quotation marks in the legitimate input showed that the Select-Fix dataset obtained a small increase reducing the false negative rate to zero. Unfortunately the already perfect classification of the Insert-Fix set was thrown off by the higher dimensional features resulting in the generation of false negatives. Therefore, the multidimensional feature extraction boosted three of the four sets but resulted in a loss of accuracy for the fourth.

The largest increases in detection accuracy occurred when the coefficients shared signs. In these solutions the higher dimensional features constructively interact to generate similarity values that reflect these features and as a result assist in the classification of the query strings. It would be computationally extremely difficult to probe the full set of coefficient combinations for higher subsequence sizes. Therefore, to test these higher subsequence lengths, the coefficients will be set to one so they are constructively interacting. This is not necessarily the best case and in the previous experiment it was seen that whilst all the coefficients shared the same sign, they didn't necessarily share the same value for maximum detection accuracy. However, this solution should be sufficient to determine the enhancement to detection accuracy produced by using larger scale features.

For this experiment, each dataset was tested with the following conditions. The gap decay factor was set to 0.0001 as it was in the previous experiment. The maximum subsequence length was tested for every integer value from one to ten. All the coefficients of the scale one to ten features was set to one. The first dataset tested was the full Select dataset. The detection accuracy of the dataset quickly rises when higher dimensional features are used in the similarity evaluations. However, the accuracy quickly peaks at $p = 2$ and $p = 3$ with a massive decrease in false positives without an increase in false negatives. Unfortunately, extending to higher dimensional features then causes the rate of false negatives to increase decreasing the detection accuracy. This experiment cannot guarantee that there are not solutions at these higher subsequence lengths that will further increase accuracy but it appears that superior results are being derived from features of length two or three characters in size. This is likely due to the SQL commands and injection statements being short substrings of this length. Figure 12 shows a plot of detection accuracy against maximum subsequence length showing the rapid peak at $p = 2$ and $p = 3$ before detection accuracy decreases at higher subsequence lengths.

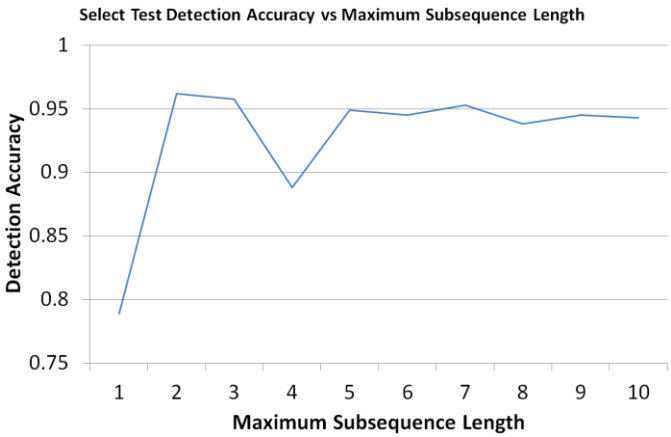


Fig. 12. A plot of detection accuracy against maximum subsequence length for the Select dataset.

This analysis was then applied to the Select-Fix dataset. The

results show that the detection accuracy decreases as higher subsequence lengths are used for classification. This is largely because the rate of false negatives increases similar to the regular Select dataset at higher subsequence lengths but as the normal queries lack confusing quotation mark input there are no false positives present for any subsequence scale. However, as was seen from experiment one applied to this dataset, there are solutions at the $p = 2$ scale that increase the detection accuracy from the ground state, only the feature scales do not have the same proportionality. Figure 13 shows the detection accuracy verses maximum subsequence length for this Select-Fix dataset.

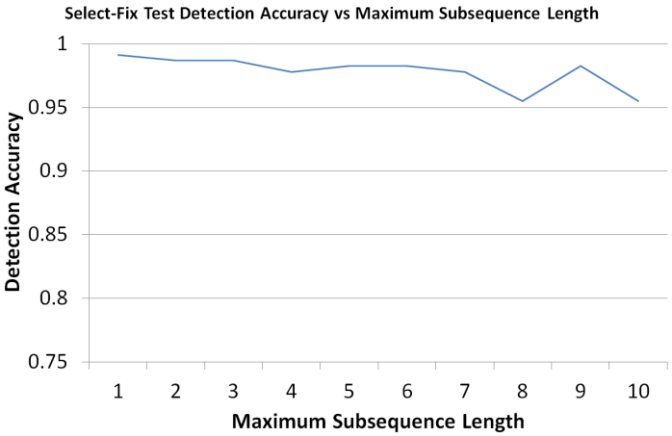


Fig. 13. A plot of detection accuracy against maximum subsequence length for the Select-Fix dataset.

Next the experiment was applied to the Insert Dataset. In the first experiment, this dataset had proven to be a lot less accurate when extended into higher dimensional features than the Select dataset. Therefore this experiment was not expected to achieve as strong an increase in detection accuracy as the Select type queries. However, the first major result is that the detection accuracy rises like the Select-type queries but instead peak at a higher value of subsequence length, specifically $p = 5$ to $p = 7$. Again, this is likely due to the primary features of the Insert-type queries being of larger length as instead of individual commands forming conditional statements, the Insert query contains a large bracket region containing the values to be entered into the database separated by commas. Despite this increase in accuracy, it comes with the cost of an increase in false negative events counteracting the large decrease of false positive events. As previously stated, false positive events are preferable to false negatives as the disruption to a service can be much greater if attack queries get through. This is especially of note since most normal legitimate queries will not be as hard to differentiate from malicious queries as this admittedly unusually difficult test dataset. The first experiment did indicate that there were higher dimensional solutions that could minimize this false negative rate by changing the proportionality of the coefficients. Therefore there are likely solutions that can maintain this accuracy but with a substantially reduced false negative rate. Figure 14 demonstrates the detection accuracy against maximum subsequence length clearly showing the peak at $p = 5$.

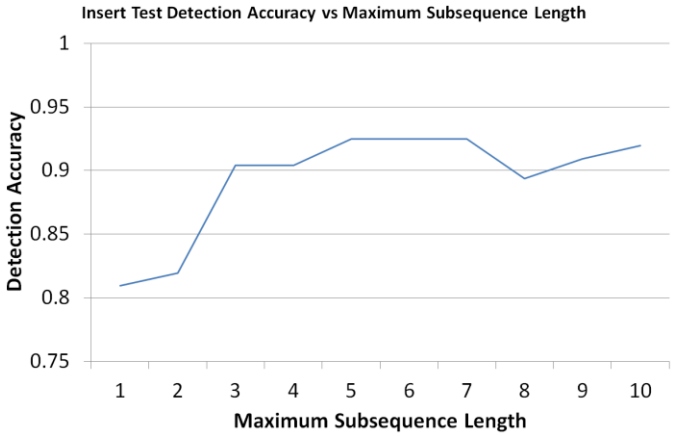


Fig. 14. A plot of detection accuracy against maximum subsequence length for the Insert dataset.

Finally, the experiment was carried out on the Insert-Fix dataset. The linear string kernel model of this dataset was of perfect detection accuracy. Combined with the first experiment showing that $p = 2$ and $p = 3$ caused a loss of detection accuracy similar to the Select-Fix dataset by an increase in false negative events with maintenance of the zero false positive events of this dataset, the Insert-Fix dataset was likely to suffer from the same issues. This was found to be true as the detection accuracy does drop from the ground state accuracy as higher subsequence length features are used in the feature extraction process.

The Gap-Weighted Subsequence Kernel is successful in reducing false positive events in confusing legitimate query strings by incorporating higher dimensional features into the similarity evaluation. Unfortunately this can be at a cost of an increase in the rate of false negative events unless an ideal combination of coefficients can be determined. Therefore it is recommended that higher subsequence lengths be used on complicated query strings that are difficult to differentiate from malicious strings but for simpler queries it is superior to limit multidimensional feature extraction to features of scales no longer than the individual SQL commands. This is due to the Feature Manipulation Algorithm alone being successful in the successful identification of all legitimate query strings allowing all unusual strings to be immediately rejected. These results indicate that the best combination of coefficients is likely to be found for values of maximum subsequence length that relate to the length of substrings within the query strings. It is worth attempting to train a model using these subsequence lengths unless the ground state detection accuracy is already perfect in which case the detection accuracy is already at the desire level and higher dimensionality will likely result in an increase in false negatives. Figure 15 demonstrates how the detection accuracy of the Insert-Fix dataset varies with maximum subsequence length.

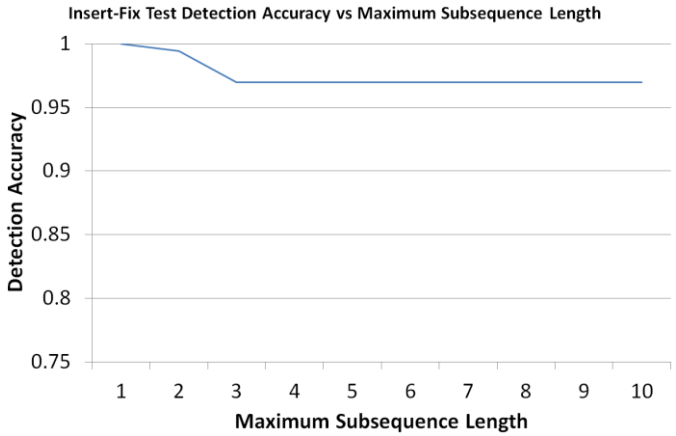


Fig. 15. A plot of detection accuracy against maximum subsequence length for the Insert-Fix dataset.

5.3. Time complexity

The two experiments discussed above were also used to generate data on the training time for the models used, the processing time for the whole Select and Insert datasets and the average processing time for each individual query in these sets. This is important for the successful operation of the proposed solution as it must be able to operate in real time in order to classify queries with little delay to the users of the defended service.

The Select and Insert training set produced a linear string kernel model with an associated training time. Then for the Select linear string kernel model, the Select and Select-Fix testing datasets were evaluated and for the Insert linear string kernel model, the Insert and Insert-Fix testing datasets were evaluated. The Select model had a training time of 2037.18 milliseconds. The Select test dataset of 232 query strings required 2443.973 milliseconds to process which produces an average processing time of 10.534 milliseconds per query. The Select-Fix test dataset of 232 query strings similar to the original dataset with the quotation marks changed to question mark placeholders took 2423.539 milliseconds to process with an average processing time of 10.446 milliseconds per query. The Insert model had a training time of 6563.565 milliseconds. The Insert test dataset of 170 query strings required 3891.915 milliseconds to process which produces an average processing time of 22.894 milliseconds per query. The Insert-Fix test dataset of 170 query strings similar to the original dataset with the quotation marks changed to question mark placeholders took 3767.71 milliseconds to process with an average processing time of 22.163 milliseconds per query.

The datasets with the quotation marks removed from the legitimate queries tended to process slightly faster due to the shorter query strings produced by the Feature Manipulation Algorithm. The Insert query model took just under three times longer to generate due to the larger set of training queries produced by the increased number of inputs into the Insert example query string. As a result, the operation time will be impacted heavily by query strings with a larger number of inputs although it is possible that the training set generator can be further refined to reduce the number of training queries required for a reliable sample size training dataset. The dimensionality of the feature vectors is always equal to the number of training queries. Despite these limitations, this experiment shows that for the linear string kernel models, the training and processing time are acceptable for runtime as the training is only required on the initialization of the demonstration software.

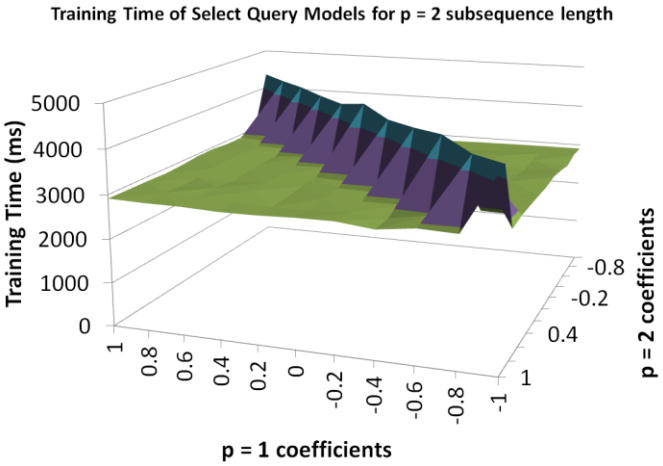


Fig. 16. A surface plot showing the training time of multiple p = 2 Select query models in milliseconds varying with the coefficient values associated with the model feature extractions.

The processing time required for higher dimension features was then measured by using the same experiments used for the detection accuracy. The different models trained for the $p = 2$ and $p = 3$ states were used to determine how the training and query processing time were influenced by the maximum subsequence length and the coefficients used for each trained model. Figure 16 shows the training time of the Select query models against the different coefficient combinations for the $p = 2$ state. The surface is flat with a large diagonal ridge. The training time is constant being independent of the coefficients of the different feature scales (except for the destructively interacting case which will be discussed shortly). This makes sense as these coefficients merely scale the results of the Gap-Weighted Subsequence Kernel algorithm and do not influence the number of calculations required. The increase in the maximum subsequence length does increase the training time required as it increases the number of multidimensional features the string kernel algorithm must iterate through. The next experiment will attempt to determine this relationship between maximum subsequence length, the training time and the query processing time.

The ridge appears to occur for values of the coefficients that destructively interact. This rapid increase in training time coincides with locations of poorest accuracy due to this interaction. Therefore it is likely that the SVM is being starved of important feature data by this interaction resulting in a poorer model. Specifically, it is likely the grid algorithm designed to determine the cost parameter that results in this increase as the feature vectors are likely heavily indistinguishable within the feature space. For the $p = 3$ state the same pattern is seen where the model training time is

independent of the coefficient values apart from the destructively interacting combinations where it substantially increases. The $p = 3$ state training time has risen again by about the same amount as the difference between the ground state and the $p = 2$ state possibly indicating that the training time varies linearly with the maximum subsequence length. The second experiment discussed shortly identifies this relationship. Figure 17 shows the surface plot produced by these $p = 3$ state model training times.

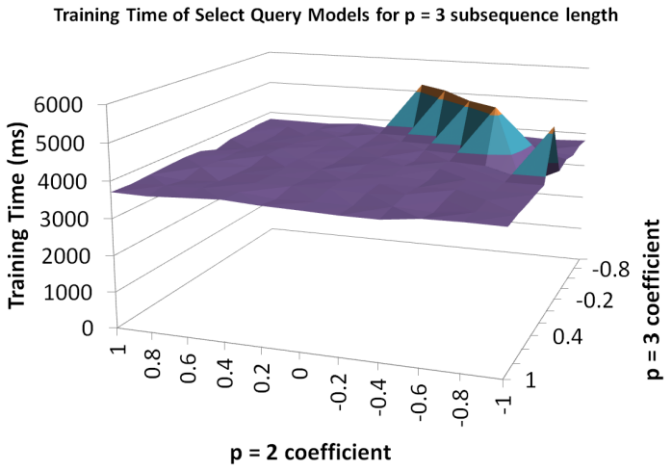


Fig. 17. A surface plot showing the training time of multiple p = 3 Select query models in milliseconds varying with the coefficient values.

The Insert query also exhibited the same features as the Select query with a ridge where the coefficients destructively interact with a flat constant training time for the other combinations. The training time again rises with the maximum subsequence length at a faster rate due to the increased size of the Insert training set. Figure 18 displays the surface produced by the training of the Insert-type query set. The amount of extra time required to generate the destructively interacting models seems to be roughly 1500 milliseconds independent of the training time of the other models indicating that they are independent of the maximum subsequence length and reinforcing the conclusion that the cost parameter grid algorithm component of the SVM is likely responsible.

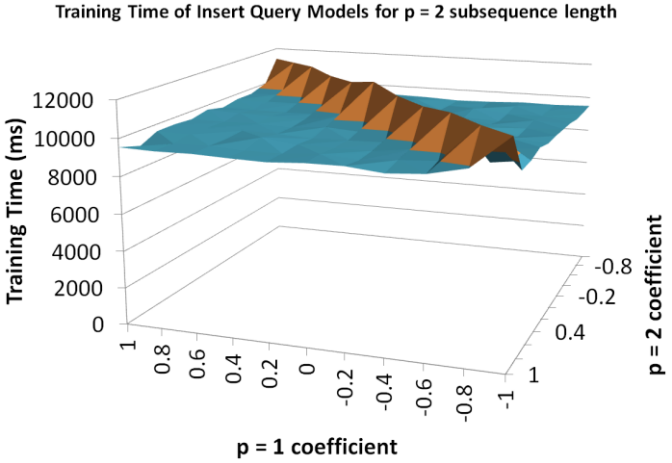


Fig. 18. A surface plot showing the training time of multiple p = 2 Insert query models in milliseconds varying with the coefficient values associated with the model feature extractions.

This experiment also recorded the processing time per query for the Select and Insert datasets based on the models produced using the different combinations of the similarity evaluation coefficients. Figure 19 demonstrates the surface plot produced by the Select dataset trained in the $p = 2$ state. When operating in the testing phase the processing time seems to mirror the training phase operation times. The processing time is again independent of the string kernel coefficients with some limited variation due to background operations of the operating system. However, in this situation the destructively interacting models process the Select dataset substantially faster than the normal models. This shorter processing time is a result of faster classification by the SVM as the feature manipulation and string kernel algorithms have the same workload with these coefficients as with any other combination. The reason behind this is not entirely understood and likely due to the SVM rejecting the model and simply applying a global malicious classification to every query within the Select dataset.

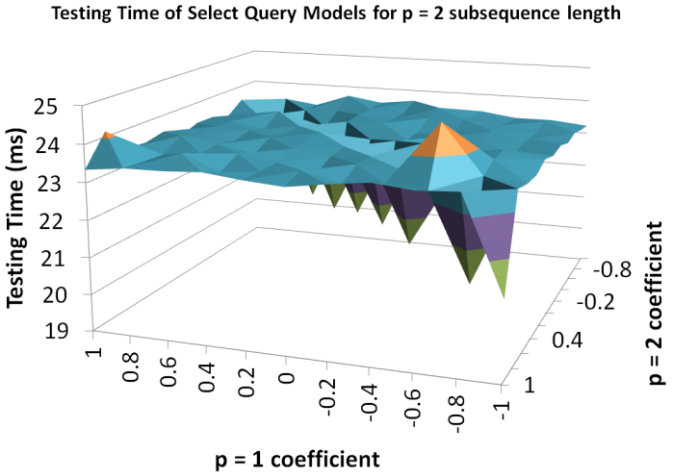


Fig. 19. A surface plot showing the per-query testing time in milliseconds of the Select dataset classified by multiple p = 2 Select query models varying with the coefficient values associated with the feature extractions.

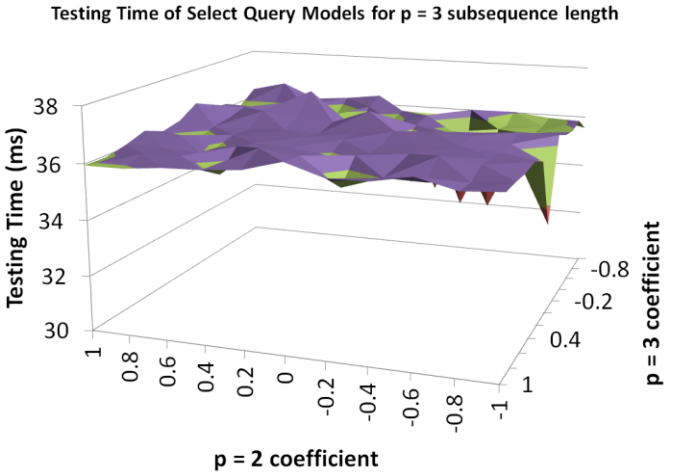


Fig. 20. A surface plot showing the per-query testing time in milliseconds of

the Select dataset classified by multiple p = 3 Select query models varying with the coefficient values.

Similar results are seen for the Select dataset operating within the $p = 3$ state where again the processing time is independent of the coefficients except for the coefficients that result in poor models where the processing time is more rapid. Additionally, as with the training times, the processing time per query appears to increase linearly with the maximum subsequence length. Figure 20 demonstrates the surface plot produced by the per query processing times of the $p = 3$ state.

Perhaps unsurprisingly, when the per query processing time of the Insert dataset is compared with the coefficients used to train the models, the same features are again seen reinforcing that the processing time is not only just independent of the coefficients but also independent of the type of query string and merely only to the length of the string. Figure 21 demonstrates the surface produced by this comparison for the Insert dataset.

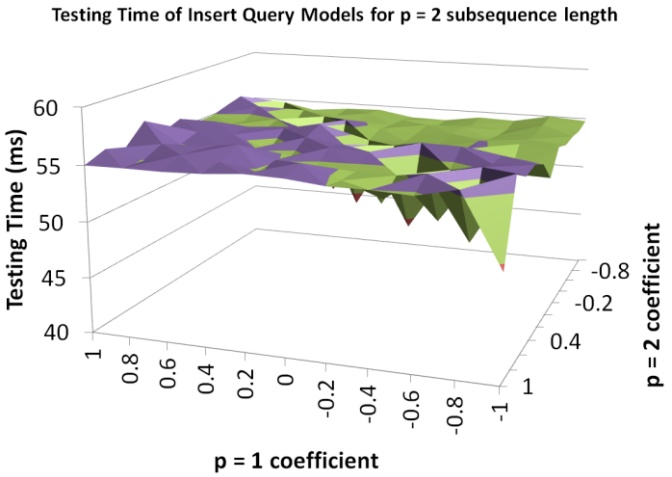


Fig. 21. A surface plot showing the per-query testing time in milliseconds of the Insert dataset classified by multiple p = 2 Insert query models varying with the coefficient values associated with the feature extractions.

The first experiment showed that the training and testing processing times were independent of the coefficients of the string kernel algorithm. Therefore the training time and processing time results from the second experiment, where the coefficients were locked to one and the maximum subsequence length was increased incrementally from one to ten, are perfect for the determination of this relationship.

Ten models were generated for the Select training set and another ten models were generated for the Insert training set. Each model had a maximum subsequence length of one to ten. The coefficients of the different scales of the similarity evaluations were set to one. The training time of these ten models was compared to the maximum subsequence length of each model to generate a plot of the relationship. An equation is then generated to determine the best fit of the points. This equation describes the relationship between the subsequence length and the training time and also describes the overhead required by other components of the program as well as the amount of time the additional iterations of higher dimensional features requires.

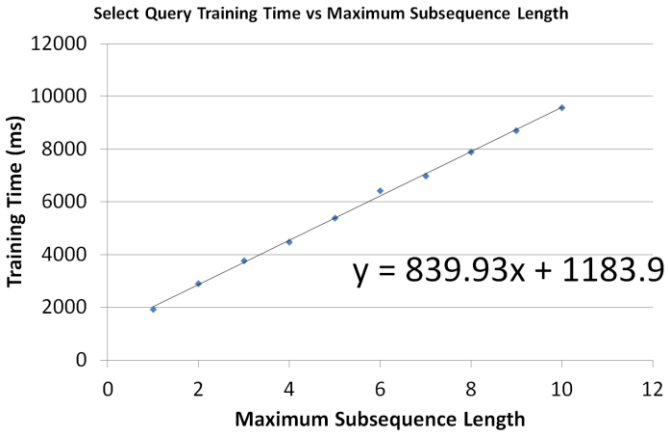


Fig. 22. A plot of model training time against the maximum subsequence length for the Select training dataset.

Applying this experiment to the Select training dataset produced the graph shown in Figure 22. As indicated by results from experiment one, the plot produced an almost perfect linear trend indicating that the relationship between maximum subsequence length and training time is linear. The linear relationship also produced an associated equation with important implications to the Select query training. The gradient of the equation indicates that for every additional extension to the maximum subsequence length, the model training time increases by 839.93 milliseconds. The intercept of the equation also shows that 1183.9 milliseconds of the training time is independent of the subsequence length and is likely due to the size of the training dataset and the operational requirements of the SVM.

Applying the same experiment to the Insert training dataset produced the graph shown in Figure 23. The Insert queries also follow this linear relationship except due to the increased size of the training dataset, both the gradient and the intercept of the linear trend are greater supporting the conclusion that it is related to the size of the training dataset and the length of the individual query strings within the training set queries as well as the operation of the SVM. The gradient of the equation indicates that for every additional extension to the maximum subsequence length, the model training time increases by 2931.9 milliseconds. The intercept of the equation also shows that 3359.6 milliseconds of the training time must be independent of the subsequence length.

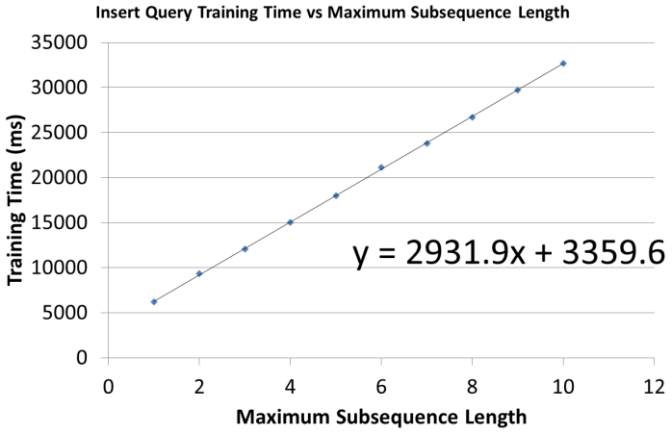


Fig. 23. A plot of model training time against the maximum subsequence length for the Insert training dataset.

Each model set was also used to test the four Amnesia-derived datasets. The Select and the Select-Fix datasets were classified using the Select models. Meanwhile, the Insert and Insert-Fix datasets were classified using the Insert models. Each classification had the associated per query processing time evaluated. By plotting these results in the same form as the training time results, it was possible to generate plots for each dataset showing how the processing time was affected by the increased maximum subsequence length.

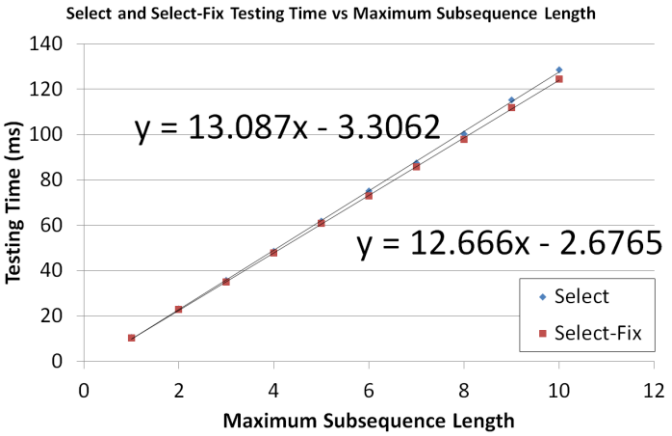


Fig. 24. A plot of the Select dataset and Select-Fix dataset processing times against the maximum subsequence length for the Select models. The top line is the Select dataset and the bottom is the Select-Fix dataset.

Figure 24 demonstrates the processing time for the queries in the Select and Select-Fix datasets using the Select models over the varying maximum subsequence lengths. The query processing times also share a linear relationship with maximum subsequence length. This is due to string kernel algorithm generating the feature vectors of the test queries. Therefore the processing time also depends on the size of the training dataset and the dimensionality of the multidimensional feature extraction. The Select dataset queries on average required an extra 13.087 milliseconds of processing time for the computation of each additional subsequence length. The Select-Fix dataset requires slightly less extra time, 12.666 milliseconds, for higher subsequence lengths. This is likely due to the output strings from the feature manipulation algorithm being shorter due to the lack of quotation marks within the legitimate queries of this testing dataset. In the plots of training time against subsequence length, the intercept was clearly not zero and was due to the operation of other algorithms during the training phase of the SVM. However, in this testing case the intercepts of the two equations are likely a result of measurement errors and should be zero intercepts as the testing phase lacks the time requirements due to the production of decision rules.

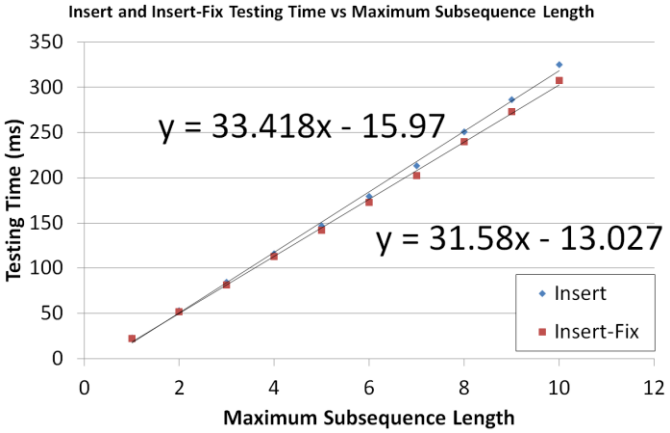


Fig. 25. A plot of the Insert dataset and Insert-Fix dataset processing times against the maximum subsequence length for the Insert models. The top line is the Insert dataset and the bottom is the Insert-Fix dataset.

This experiment was also evaluated on the Insert and Insert-Fix datasets. The results of this evaluation are displayed within Figure 25. The linear relationship also exists for the Insert queries again showing that these relationships are independent of the query type and only the length of the query strings and the scale of the multidimensional feature extraction. The increased size of the training dataset results in increased processing time due to the increased dimensionality of the feature vectors of the test query strings. The Insert dataset queries on average required an extra 33.418 milliseconds of processing time for the computation of each additional subsequence length. The Select-Fix dataset requires slightly less extra time, 31.58 milliseconds, for higher subsequence lengths. This is likely due to the same feature manipulation algorithm string length situation as the Select-Fix dataset query strings.

The detection accuracy analysis concluded that the superior subsequence length for the Select queries was $p = 2$ or $p = 3$ whereas for the Insert queries this peak in accuracy occurred at subsequence lengths of $p = 5$ to $p = 7$. The above equations combined with this information can be used to estimate the model training time and the individual query processing time for the Select and the Insert datasets. The Select query dataset training time ranges from 2863.76 milliseconds to 3703.69 milliseconds and each test query takes approximately 22.6555 milliseconds to 35.9548 milliseconds on average to process. The Insert query dataset training time ranges from 18019.1 milliseconds to 23882.9 milliseconds and each test query takes approximately 144.873 milliseconds to 217.956 milliseconds on average to process. Even the worst

case of these calculations places the program firmly within real time operation with acceptable query delay times due to this solution as the training time only applies once during software initialization. The Select queries process significantly faster than the Insert queries, most likely due to the increased input trajectories of the Insert query and the larger associated training set. Queries with significantly more input fields may cause this solution difficulty in maintaining the real time operational requirement unless the input generator algorithm can be enhanced in the future. Longer query strings will also negatively affect the processing time however, generally the most complicated strings are due to injected input therefore the user that suffers the most from query response delay is an attacker. At the very least this might discourage the attacker from making the attempt if they must craft many attack queries.

5.4. Spatial complexity

The memory consumption of the software during use is also of importance as it must be capable of operating on the web server hosting a web application with minimal system impact. On start up the program required 16 Mb of memory. The Input Generator algorithm placed another 2.5 Mb of demand on the system memory resources. Training a model required 6.5 Mb of memory. This memory requirement was independent of the maximum subsequence used for the training phase. The testing of the 232 queries within the Amnesia-derived Select dataset required an additional 10 Mb of memory. The demonstration software had a peak memory draw of only 35 Mb when using the Select example query. The memory requirements of this solution border on negligible and therefore many instances can be run simultaneously to defend multiple web application queries on a single machine.

Despite the scale of calculations being performed to determine the multidimensional feature set and compute the similarity values of these features, the software stores very little of the results of these calculations within memory. Therefore the requirements are kept minimal as the only values that are kept through the iteration through the full set of multidimensional features are the cumulative inner product similarity evaluations. More memory is required by larger training datasets as the feature vectors, stored in memory for each test query, are of larger dimensionality. Despite these requirements, the demonstration software still maintained the peak memory draw of 35 Mb when using the Insert example query training a model of high dimensionality to classify the Amnesia-derived Insert dataset. This is one of the strongest advantages of this proposed solution. Hundreds of thousands of potential features can be evaluated by the similarity algorithm and used to train models and test query strings but the string kernel algorithm does not require any of these features to be stored in memory. The algorithm simply iterates through the multidimensional feature space of hundreds of thousands of features yet generates a feature vector only two to three digits in dimensionality greatly limiting the pressure placed on the system resources of the server running software utilizing this method.

VI. Discussion

The proposed solution runs on a web server as the web application sends generated queries to a source IP and port hosted by the solution. The solution is then able to test queries and then relay them to the back-end database server as well as pass the reply messages directly to the web application with minimal latency. In essence, the detection solution is transparent to the communications between the web application and the back-end database except for the outbound communications where they are delayed by a number of milliseconds for the classification process.

The software was required to have high detection accuracy with a low rate of false positives and false negatives combined with a processing time rapid enough for real time operation. By using the Gap-Weighted String Subsequence Kernel algorithm to compute the inner product of multidimensional features, solutions were found that improved detection accuracy over a linear approach of simple features in the query strings. As the operation of this algorithm depended on the maximum subsequence length, the gap decay factor and a set of coefficients for features of length one character to the maximum subsequence length, these inputs were evaluated to determine the superior solution for detection accuracy. The models generated by the program were heavily influenced by the maximum subsequence length and its associated coefficients. However, the gap decay factor was not a major component in the accuracy of the generated models.

In the process of classifying SQL Injection attacks, the solution with the highest detection accuracy may not be the best as, whilst the detection accuracy places no bias between false positives and false negatives, in reality false negatives are significantly less desirable than false positives. False positives result in service disruption whereas false negatives can result in service destruction. The solution is capable of identifying all the different types of SQL Injection Attack except for Stored Procedures as the query string cannot be intercepted at the web server. Despite this weakness, it might be substantially easier to update the code of a stored procedure to sanitize the inputs compared with the updating of third party software in the form of the web applications and therefore it might be acceptable to struggle to identify stored procedure attacks.

Much of this difficulty in classifying the datasets was a result of unsanitised quotation marks within the testing datasets. These were introduced to test the maximum tolerances of the solution. Copies were produced of the Select and Insert datasets that extracted quotation marks from the legitimate queries as quotation marks are unlikely to be seen during normal operation of these queries. This allowed the proposed solution to achieve an accuracy of 100% on both test sets. Whilst this appears to be advantageous, the multidimensional feature extraction that this proposed method utilizes was only required for the Select queries as the Insert queries actually achieved 100% with the ground state accuracy. Additionally, for these testing datasets with high ground state accuracy, the multidimensional analysis on higher dimensional features could result in an inferior model

from the simple feature extraction. Therefore this solution, whilst competent on simple query strings, shows its true capability with complicated query strings which are hard to differentiate due to sharing similarities to attack strings during legitimate operation.

The proposed solution's processing time was also evaluated to determine if the real time operation condition was fulfilled. The time complexity of the solution obeys the equation $t = I \times p \times N$ where I is a constant determined by the processing power of the available CPU, p is the maximum subsequence length and $N = |u| \times |u| \times n$ where $|u|$ is the average character length of the training set input strings and n is the total number of query strings within the training set. This relationship was predicted by the order of the Gap-Weighted String Subsequence Kernel dynamic processing algorithm that exhibited a time complexity of the order $O(p|s||t|)$ where p is the maximum subsequence length, $|s|$ is the length of the first argument string and $|t|$ is the length of the second argument string. The processing time of the other algorithms is of negligible time compared to the string kernel algorithm and therefore do not contribute to the relationship.

As the time complexity is heavily dependent on the number of training set queries, the size of this set must be limited to produce enough information about the query generation of the web application without being overly descriptive. The Feature Manipulation Algorithm shrinks the length of the testing strings reducing the processing time of the string kernel algorithm. Unfortunately, the Input Generator algorithm produces significantly larger training datasets with queries containing many input fields. These larger sets then result in substantially increased processing times. The Input Generator is therefore a target for further improvements by minimizing the training set it generates whilst still maintaining an acceptable sample dataset for the complete set of possible queries.

VII. Conclusion and Future Work

Our proposed method has more success with complicated query strings compared to the simple strings. Fortunately, despite the weaknesses associated with simple attacks, most of these simple strings have already been identified and documented. A major danger is more sophisticated novel attacks that have yet to be encountered. However, these more sophisticated attacks tend to result in longer and more feature-rich query strings and, as a result, are very quickly detected by the proposed solution. None of the complicated attack strings caused by more sophisticated attacks such as Inference attacks failed to be correctly classified and blocked by the demonstration software. The training set did not directly describe features within these sophisticated attacks, but the deviation from the legitimate strings was enough to warrant a malicious classification. This property of the solution allows it to be adaptable to attacks not explicitly described in the training set, both discovered and undiscovered. The difficulty is focused on the successful description of simple attacks within the training dataset.

The successful identification of the ideal coefficients for weighting the features of the string kernel algorithm is extremely important. The number of possible coefficients for higher dimensional solutions rises exponentially making the discovery of the optimum set of coefficients non-trivial. As the superior solutions are limited to smaller maximum subsequence length, the coefficient combinations are not overly large in size during normal operation. The difficulty of this process can be managed through the realization that the values of the coefficients are not the important factor but instead the proportionality between the different coefficients is the mechanism that alters the accuracy of the trained models.

As the demonstration software only showed a proof of concept of the design algorithms, the next milestone would be to deploy the solution as an actual defensive module for a web application and database server. This would require the implementation of the full Input Generator algorithm. This algorithm would be required to identify the possible input trajectories for a web application and determine how many different output queries are produced and how they are related to these inputs. The algorithm would then be able to open a number of new threaded operational modes for each query type to generate a series of models.

Most of the time complexity of this proposed solution lies in the computation of the Gap-Weighted String Subsequence Kernel function. Additionally, there is evidence indicating that the models being produced by the SVM have a high variance which results in models that suffer from overfitting the training dataset. Both these weaknesses can be compensated by decoupling the number of 'landmark' strings from the number of the training set strings. Currently, this coupling results in an exponential increase in the number of kernel function operations and therefore training and per-query processing time, upon the increase of the training dataset size. If a fixed-length set of 'landmark' strings can be identified that result in accurate models, these strings could replace the training set strings in the computation of the feature vectors of the query strings.

Additionally, this decoupling would allow for the increase in the training set size with impact only to the model training phase processing time and not the per-query processing time which is the more critical time-dependent component. Increasing the size of the training set is a well-documented method of reducing the variance of machine learning models. As a result, the solution could be made truly intelligent by the allowing it to incorporate new query strings previous models misclassified into a new training set to be trained into an improved model when processing power is available during downtime.

Currently the proposed solution depends on $p + 2$ variables. However, if the decoupling operation described above is successful, it may be possible to output the individual subsequence length kernel function evaluations $K_q(x_i, x_j)$ as features instead of relying on a weighted sum of these values up to the maximum subsequence length. Limited testing of this method indicates that a set of parameters (related to the

old coefficients) might still be needed. This would reduce the number of inputs to just two variables, the maximum subsequence length and the gap decay factor. It would be ideal if the software could employ a form of optimization algorithm to identify the values of these two variables that minimize the error of the classification. A second optimization algorithm that can determine the parameter set that minimizes the classification error. If these two algorithms could be optimized simultaneously, the superior classification model could be generated automatically. It is of note that this optimization process would likely heavily influence the time required to train models but fortunately, would have no effect on the testing phase processing time.

ACKNOWLEDGMENT

This work was conducted through Liverpool John Moores University in partial fulfilment of the requirements for the degree of Masters in Computing and Information Systems. The Microsoft Visual Studio 2013 integrated development environment software was supplied through Microsoft DreamSpark. The LibSVM java library was supplied freely online by Chih-Chung Chang and Chih-Jen Lin. Version 2.89 of this java library was translated into a clean .NET library by Matthew Johnson.

REFERENCES

Z. Su and G. Wassermann, "The Essence of Command Injection Attacks in Web Applications," In the 33rd Annual Symposium on Principles of Programming Languages (POPL2006), 2006.

W. G. J. Halfond, J. Viegas and A. Orso, "A Classification of SQL Injection Attacks and Countermeasures," In proceedings of the International Symposium on Secure Software Engineering 2006.

J. Williams and D. Wichers, "Top Ten Most Critical Web Application Vulnerabilities," 2013. [Online]. Available: https://www.owasp.org/index.php/Top_10_2013-Top_10. [Accessed 1 August 2015].

Trustwave. Trustwave 2015 global security report. https://www2.trustwave.com/rs/815-RFM-693/images/2015_TrustwaveGlobalSecurityReport.pdf; 2015 [accessed 18.05.15]

H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini and C. Watkins, "Text Classification using String Kernels," Journal of Machine Learning Research, no. 2, pp. 419-444, 2002.

C. Cortes and V. Vapnik, "Support-vector Networks," in Machine Learning, 3 ed., vol. 20, Springer, 1995, pp. 273-297.

A. Liu, Y. Yuan, D. Wijesekera and A. Stavrou, "SQLProb: a proxy-based architecture towards preventing SQL injection attacks," SAC 2009, 2009.

M.-Y. Kim and D. H. Lee, "Data-Mining based SQL injection attack detection using internal query trees," Expert Systems with Applications, vol. 41, no. 11, pp. 5416-5430, 1 September 2014.

G. T. Buehrer, B. W. Weide and P. A. G. Sivilotti, "Using parse tree validation to prevent SQL injection attacks," In proceedings of the International Workshop on Software Engineering and Middleware (SEM) at Joint FSE and ESEC, pp. 106-113, 2005.

I. Lee, S. Jeong, S. Yeo and J. Moon, "A novel method for SQL injection attack detection based on removing SQL query attribute values," Mathematical and Computer Modelling, vol. 55, pp. 58-68, 29 January 2011.

F. Valeur, D. Mutz and G. Vigna, "A Learning-Based Approach to the Detection of SQL Attacks," In Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA), 2005.

I. Homoliak, "Increasing Classification Accuracy in LIBSVM using String Kernel Functions," Student EEICT, Volume: 2, 2012.

J. Rouso and J. Shawe-Taylor, "Efficient Computation of Gapped Substring Kernels on Large Alphabets," Journal of Machine Learning Research, vol. 6, pp. 1323-1344, 2005.

C.-C. Chang and C.-J. Len, "LIBSVM: A Library for Support Vector Machines," ACM Transactions on Intelligent Systems and Technology, vol. 2, no. 3, p. article 27, 2007.

W. G. J. Halfond and A. Orso, "AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks," In Proceedings of the IEEE and ACM International Conference on Automated Software Engineering (ASE 2005), 2005.

P. Bisht, P. Madhusudan and V. N. Venkatakrishnan, "CANDID: Dynamic candidate evaluations for automatic prevention of SQL Injection Attacks," ACM Transactions on Intelligent Systems and Technology, vol. 13, no. 2, p. 14, 2010a.

P. Bisht, T. Hinrichs, N. Skrupsky, R. Bobrowicz and V. N. Venkatakrishnan, "NoTamper: Automatic Blackbox Detection of Parameter Tampering Opportunities in Web Applications," In Proceedings of the 17th ACM Conference on Computer and Communications Security, p. 272-288, 2010b.

B. Kar, S. Panigrahi and S. Sundararajan, "SQLiGoT: Detecting SQL Injection attacks using graph of tokens and SVM," Computers and Security, vol. 60, pp. 206-225, 2016.

H. Shahriar, S. North and W. Chen, "Client-Side Detection of SQL Injection Attack," In: Advanced Information Systems engineering workshops, Springer, p. 512-517, 2013.

A. Moosa, "Artificial Neural Network based Web Application Firewall for SQL Injection," World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering, vol. 4, no. 4, 2010.

A. Makiov, Y. Begriche and A. Serhrouchni, "Improving web application firewalls to detect advanced SQL Injection attacks," In: Information assurance and security (IAS) 2014 10th International conference on, IEEE, p. 35-40, 2014.

K. Zhang, C. Lin, S. Chen, Y. Hwang, H. Huang and F. Hsu, "TransSQL: A translation and validation-based solution for SQL-Injection attacks," In: Robot, Vision and Signal processing (RVSP), 2011 first international conference on, IEEE, p. 248-251, 2011.

M. Le, A. Stavrou and B. B. Kang, "DoubleGuard: Detecting Intrusions in Multitier Web Applications," IEEE Transactions on Dependable and Secure Computing, vol. 9, issue 4, p. 512-525, 2012.

C. I. Pinzón, J. F. De Paz, Á. Herrero, E. Corcado, J. Bajo and J. M. Corchado, "idMAS-SQL: Intrusion Detection Based on

MAS to Detect and Block SQL Injection through data mining," Information Sciences, vol. 231, pp. 15-31, 2013.
S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," Journal of Molecular Biology, vol. 48, issue 3, pp. 443-453, 1970.

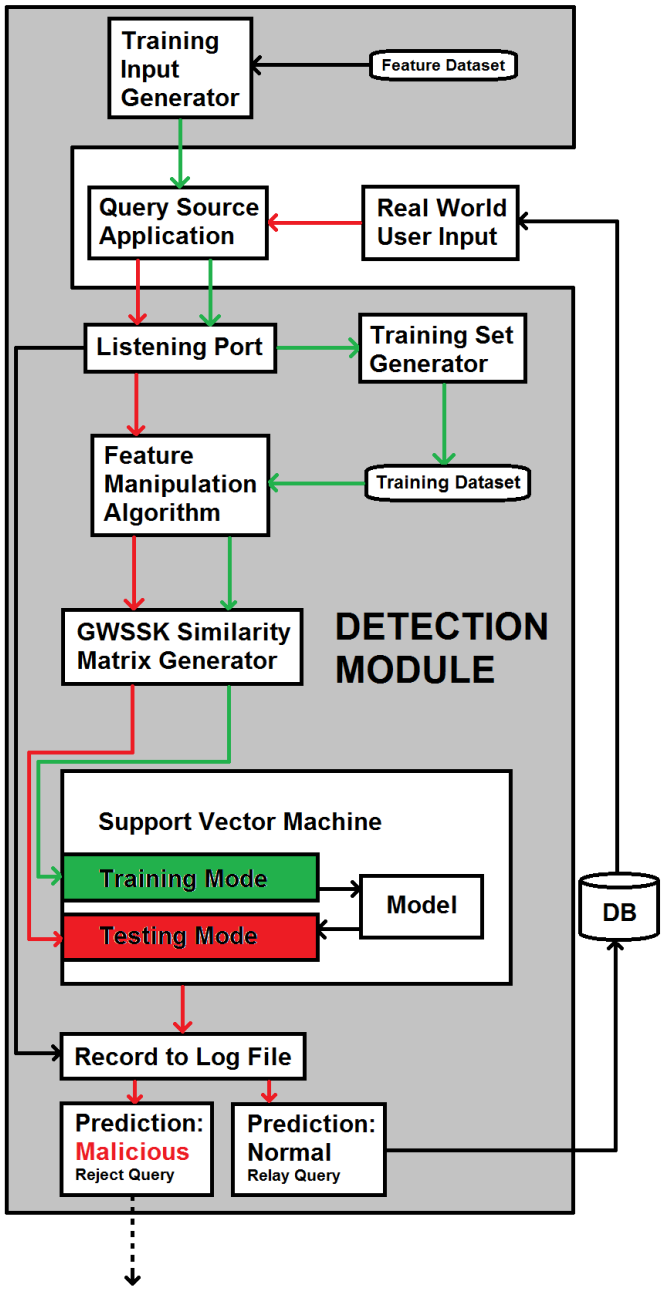


Fig. 26. The SQLIA detection framework.