An Applied Pattern-Driven Corpus to Predictive Analytics in Mitigating SQL Injection Attack

Solomon Ogbomon Uwagbole
School of Computing
Edinburgh Napier University
Edinburgh, United Kingdom
05012238@live.napier.ac.uk

William J. Buchanan, Lu Fan
School of Computing
Edinburgh Napier University
Edinburgh, United Kingdom
b. buchanan; l.fan@napier.ac.uk

Abstract— Emerging computing relies heavily on secure back-end storage for the massive size of big data originating from the Internet of Things (IoT) smart devices to the Cloud-hosted web applications. Structured Query Language (SQL) Injection Attack (SQLIA) remains an intruder's exploit of choice to pilfer confidential data from the back-end database with damaging ramifications. The existing approaches were all before the new emerging computing in the context of the Internet big data mining and as such will lack the ability to cope with new signatures concealed in a large volume of web requests over time. Also, these existing approaches were strings lookup approaches aimed at on-premise application domain boundary, not applicable to roaming Cloud-hosted services' edge Software-Defined Network (SDN) to application endpoints with large web request hits. Using a Machine Learning (ML) approach provides scalable big data mining for SQLIA detection and prevention. Unfortunately, the absence of corpus to train a classifier is an issue well known in SQLIA research in applying Artificial Intelligence (AI) techniques. This paper presents an application context pattern-driven corpus to train a supervised learning model. The model is trained with ML algorithms of Two-Class Support Vector Machine (TC SVM) and Two-Class Logistic Regression (TC LR) implemented on Microsoft Azure Machine Learning (MAML) studio to mitigate SQLIA. This scheme presented here, then forms the subject of the empirical evaluation in Receiver Operating Characteristic (ROC) curve.

Keywords— SQL Injection; SQLIA Data analytics; SQLIA Pattern-driven data set; SQLIA big data; SQLIA hashing

I. Introduction

Recent years have seen a continuous upward trend in big internet data, and the volume of the Cloud-driven applications will only continue to grow with more individuals, governments and businesses adopting and hosting files and applications in the Cloud. A Google search of 'SQLi hall of shame' [1] throws light on how topical SQLIAs issues are. SQL Injection (SQLI) is not only a vulnerability arising from developers' lack of security awareness in web application development to sanitised input data, but an exploit of the free text processing capability of the SQL engine which has ramifications in both legacy and new web application lacking sanitation becoming SQLI vulnerable.

The SQL language syntax is in plain English, and the SQL keywords and tokens are also in plain text. Therefore, the SQLIA problem is a plausible candidate to apply predictive analytics employing a supervised learning model trained with historical attack signatures, including SQL tokens and safe web requests patterns to predict SQLIA at SQL query injection points.

Unfortunately, web applications are designed for different requirements. Thus, web application domain context is so diverse to have a standardised pre-existing pattern-driven data set to train a supervised learning model. Moreover, as such, non-availability of a data set that covers every application domain context is an issue well known in SQLIA research in applying AI techniques.

We opine patterns exist in every input data in both legacy and new web applications that can be leveraged to generate as many derivations of member strings. Applying ML techniques requires data set with sufficient learning data arising from patterns that exist in the input data. There is a need to build a prediction model trained from data set of the desired application domain context to predict SQLIA. In our labelling of the data set, the presence of known attack signature at injection points will contain patterns of SQL tokens and symbols which are deemed SQLIA positive. Conversely, the valid web requests (SQLIA negative) would take the form of generating all possible member strings. We apply Non-Deterministic Finite Automata (NFA) implemented in Regular Expression (RegEx) to define the constraint patterns, and employing Symbolic Finite Automata (SFA) with a constraint solver termed Satisfiability Modulo Theories (SMT-Z3) [2], [3] to generate member strings from the defined RegEx patterns.

We trained a supervised learning model with this pattern-driven learning data in demonstrating a proof of concept by applying predictive analytics to a test web application expecting dictionary word list as input data. The learning data are obtained by automata states walk to derive as many member strings in a given pattern. The trained models are evaluated in ROC curve with the TC SVM having the best performance metrics in Area Under Curve (AUC) value of *0.986* deployed as a Web Service (WS). The WS is consumed in a Fiddler proxy Application Programming Interface (API) [4] for the ongoing SQLIA prediction as to reject intercepted requests that are positive.

This paper lays out in six sections ending with a conclusion and future work summary. Section II covers related work and Section III focused on the background, including corpus generation; with Sections IV and V are detailing predictive analytics experiment, evaluation and results.

II. Related Work

The research area of SQLIA has seen various methodologies proposed over the years by researchers which we broadly categorised into three groups. Firstly, SQLI Vulnerability (SQLIV) testing and detection [5], [6]. Secondly, defensive coding in web application code sanitation for SQLI prevention [7], [8]. Thirdly, dynamic runtime analysis [9], [10] including taint-based [11], [12] and approaches that apply AI (a similar approach implemented in this paper) in the detection and prevention of SQLIA.

The approach presented in this article applies AI which requires a robust data set with various patterns in feature values to train a classifier. Applying ML requires robust learning data with patterns to train a classifier implementing TC SVM and TC LR algorithms to predict SQLIA accurately. Unfortunately, as there is no unified pre-existing data set, researchers over the years have resorted to various approaches in generating sample data sets with most proposals lacking patterns to enhance learning data, and fraught with complex computational overheads. Also, plausible approaches in the past that relied on source code will be inapplicable over time in emerging computing platforms like the Cloud where source code access for static scanning is restricted. Finally, some existing ML implementation lack completeness in being theoretical or not moving beyond the performance metrics or ROC curve evaluations to implement the trained models in a real-life application. Below gives a high-level review of some existing ML approaches.

Bockermann et al. [13] proposed using tree kernels for analysing SQL statements in addition to exploring feature vectorization of data input to an SVM classifier but found there to be drawbacks in the tree-kernels computational overhead. Also, their data set extraction depends on URL strings which are repeating strings lacking patterns.

Komiya [14] proposed SVM to detecting SQLIA from user inputs extracted from blank separation (counting number of terms) and tokens. Its drawback was requiring access to source code in detecting SQLIA.

Choi et al. [15] train an SVM classifier using feature vectorization by N-Grams. The data set is a collection of full query structure vectorised to N-Grams, but the training data is not robust enough for a good performance metrics in big data mining. Also, it needs access to full source code.

Wang and Li [16] proposed SQL query program tracing in which related queries are grouped based on runtime program trace. However, the approach's drawback is the reliance on the source code for the SQL tracing grouping that is hashed to the vector matrix for a classifier implementing the SVM algorithm.

Pinzón et al. [17] present a multi-agent approach that uses both SVM and neural networks to predict SQLIA from SQL queries behaviour that are stored as cases. The architecture named CBR cycle is computationally expensive and need access to source code.

Kim and Lee [18] proposed an approach that uses the SVM classifier for a binary classification of internal query representation known as query trees. The training data comprised of feature vectors of transformed query trees of the internal query structure. It drawbacks in requiring access to the source code and it is computational complex because of the size of the query trees.

The scheme presented here is an improvement on a previous workshop paper on applying ML predictive analytics to SQLIA prediction and prevention [19]. We detail in this article; the novel data set generation and a further improvement of the trained model's prediction results on True Positives (TP) and True Negatives (TN), but with low False Positives (FP) and False Negatives (FN). The methodology adopted here has been empirically evaluated in the numeric encoding of both expected input and patterns of SQLIA types [20], [21]. The good results of the encoded patterns-driven data set to a supervised learning model in performance metrics motivated this paper.

We present in this article a supervised learning model that uses a data set input from patterns of expected input data, including SQLIA types and SQL keywords to train various classifiers with a better performance metrics trained model deployed as WS. The scheme relies on the intercepted input data at runtime to detect and prevent SQLIA as against static queries comparison including source code scanning for vulnerabilities (white-box and black-box penetration testing) as proposed in most existing approaches applying runtime analysis.

III. Background Theory

The scheme presented here uses a web proxy to intercept web requests of any intent and applies predictive analytics techniques to predict SQLIA at the SQL injection points.

A web proxy is the most suitable to intercept web requests, including those originating from any injection mechanisms to SDN Cloud applications' endpoints. An application-level proxy performs better in intercepting and decrypting of obfuscated web requests than low-level network packet interception tools which suffer from messages fragmentation in a large volume of gigabits per seconds of packets in the wire. Injection mechanisms to a vulnerable application can originate from web page forms, second-order injection, exploiting web-enabled server variables, query strings, and through cookies.

An intruder would employ the following SQLIA types techniques to carry out the attack at the injection points in any combination. These SQLIA exploits techniques are Tautology; Union; Piggybacked; Invalid/Logical queries; Time-based; Obfuscation encoding and Stored procedure. The SQLIA types are also a source of SQLIA positive labelled feature values in the scheme presented here.

A. SQL Language structure and injection point

SQL element comprises of tokens which are labelled SQLIA positive in the data set discussed in this article. SQL tokens comprise of keywords that include identifiers, operators, literals and punctuation symbols. The SQL syntax language element has the following: SQL Clause (WHERE, SET, UPDATE, etc.); predicate (as in *LoginName = 'bob'*); and, expression (as in *'bob' OR 2=2*) which is illustrated in Fig. 1 below.

The presence of SQL tokens in web requests' expected input data when the SQL query injection point is analysed is predicted as SQLIA. In a SQL query, the WHERE clause predicate and the expression used to control query results are the SQL injection spots. A malicious query string can be passed to a SQL expression in tautological SQLIA type (e.g., *'x'='x' OR 2=2*) to return results from a table far beyond the developer's intention. This location has been explored in SQLIA research including SQLProb by Liu et al. [22] in detection and preventing SQLIA.

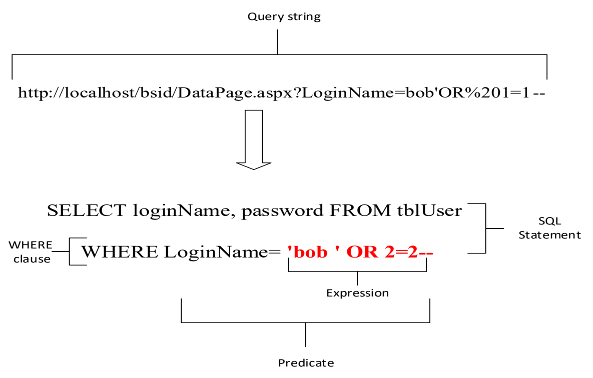

Query string
http://localhost/bsid/DataPage.aspx?LoginName=bob'OR%201=1 --
SELECT loginName, password FROM tblUser
WHERE LoginName= 'bob ' OR 2=2--
SQL Statement
WHERE clause
Expression
Predicate

Fig. 1. Query string, SQL query element and injection hotspot.

B. *Obtaining the learning data (corpus)*

We explore automata states walk to generate a data set of patterns of expected input data where none exists to train a supervised learning model. Also, a pattern-driven approach prevents the security implications of making the learning data input to ML being a repository that lay bare the expected input data.

Fig. 2 below illustrates the fundamentals of Finite State Automata (FSA) [23], [24] states walk that forms the building block to our learning data extraction techniques. These automata states walk collation to generate all possible accepted member strings shown in Table I is automated, employing a utility named Regular expression explorer (Rex) [3] by Veanes et al. [26]. Rex is an implementation of SFA which uses Z3 [25] constraint solver [2] to generate as many member strings as possible from a defined RegEx constraint patterns. In Rex, the RegEx implements an NFA with an epsilon move and a further conversion of NFA to a Deterministic Finite Automata (DFA) for optimisation.

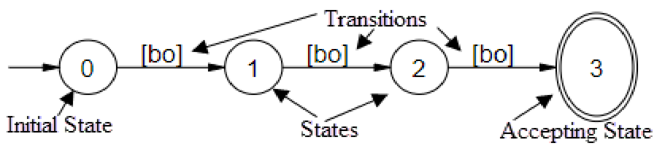


Fig. 2. States walk from expected input string to generate member strings.

TABLE I. TRANSITIONS / STATES WALKS INTERPRETATION

	Transitions			Accepted member strings
	0,1	*1,2*	*2,3*	
Alphabet Σ= *{b,o}*	b	o	b	bob
	b	o	o	boo
	o	o	b	oob
	o	o	o	ooo
	o	b	o	obo
	b	b	b	bbb
	b	b	o	bbo
	o	b	b	obb

To simplify the presented derivation techniques of generating the member strings, we use a string *"bob"* throughout this section. The process can be replicated as many feature values as desired in the intended data set. It also must be pointed out that the context of this paper is big data scenario with a large volume of feature values of strings, numeric and alphanumeric in nature that a normal string signatures lookup will not be scalable. The string *"bob"* is a minute representation for the simplification of the big picture in the significant learning data generation.

The RegEx pattern input file to Rex command line is written as: *Rex /r: InputFile.csv /k: 8* where *r* and *k* are the source input file and the size command options to generate member strings respectively. The input file would contain patterns around original strings and size where exist, e.g. *^[bob]{3}$*. Alternatively, where there are no precursor strings, RegEx patterns are inferred from the structure and size of the expected input data, e.g. *(^[b](?:[a-z]{2})$)*. These methods are described in the subsequent sections below.

1) RegEx pattern from strings and size of dictionary word list: A string *S* is a finite combination of symbols *{b,o}* that is extracted from the alphabet which is denoted by Σ. Therefore, the string *"bob"* with symbols is expressed as alphabet $\Sigma = \{b, o\}$. The Language *L* is a set of strings with a defined length denoted by $|S|$. In this example, the string *"bob"* has length three represented as $|S| = 3$. Applying Kleene Closure or Plus denoted by Σ^+ then, $\Sigma^+ = \Sigma^* - \varepsilon = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \cup \ldots_n$ where $\Sigma^* - \varepsilon$ is the Kleene Star (Σ^*) minus epsilon or an empty string (ε), $\Sigma_{1..n}$ are the finite sets of possible member strings with the length that can be generated. Therefore if alphabet $\Sigma = \{b,o\}$,$|S|=3$, then the accepted member strings is expressed as $\Sigma^+ = \{bob, boo, oob, ooo, obo, bbb, bbo, obb\}$ as shown in Fig. 2 collated in Table I.

2) RegEx pattern from strings structure and size of expected input data: We apply RegEx to produce constraints of patterns that exist in an expected input data passed to Rex utility to derive member strings. The RegEx pattern *(^[b](?:[a-z]{2})$)* accept strings that start with symbols *b* with any combinations of symbols *[a – z]* where string length $|S| = 2$, then the total string length is $|S| = 3$. Therefore, $\Sigma^+ = \{bhz, bwc, bdy, bsj, blm, bzc, bam, bby\}$ are accepted member strings as shown in Fig. 3 automata states walk diagram.

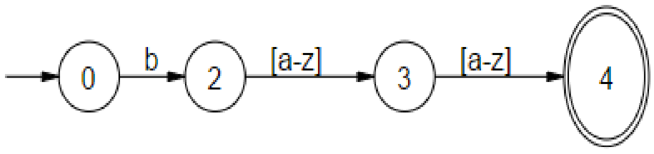
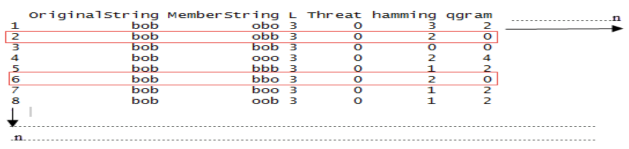


Fig. 3. States walk from RegEx pattern to generate member strings.

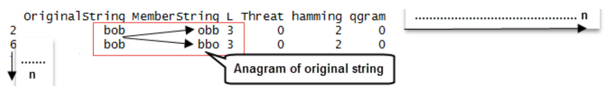
3) Member strings transposition: We further measure the string distance to compare the pattern of the original string with derived member strings by string transposition as to filter the anagram patterns of the generated member strings. We explore R stringdist (string matching package in R) [27] in the member string transposition to improve the performance of the data set in a binary classifier including an intruder attempt to circumvent the classifier by transposition (shuffled strings). Table II is a snippet of the original string *"bob"* with the derived member strings distance values. Table III illustrates string distance patterns measure by comparing Hamming *H* and qgrams *Q* to obtain the transposed (anagram) member strings [27].

TABLE II. STRING DISTANCE MEASURES BETWEEN ORIGINAL AND DERIVED MEMBER STRINGS



	OriginalString	MemberString	L	Threat	hamming	qgram
1	bob	obo	3	0	3	2
2	bob	obb	3	0	2	0
3	bob	bob	3	0	0	0
4	bob	ooo	3	0	2	4
5	bob	bbb	3	0	1	2
6	bob	bbo	3	0	2	0
7	bob	boo	3	0	1	2
8	bob	oob	3	0	1	2

TABLE III. FILTERED FEATURE VALUES CONTAINING TRANSPOSITION OF ORIGINAL STRING WHEN *H*= *0* AND *Q* *!= 0*



	OriginalString	MemberString	L	Threat	hamming	qgram
2	bob	obb	3	0	2	0
6	bob	bbo	3	0	2	0

Anagram of original string
....
n
n

4) SQLIA labelling: The derived member strings as detailed above is labelled SQLIA negative (*0*) while the presence of SQL tokens, symbol and existing known SQLIA signatures during member strings preprocessing is labelled SQLIA positive (*1*). The binary class of *0* or *1* is to be predicted at SQL query injection points. The learning data labelling routine in R language is illustrated in the Fig.4 flow chart below.

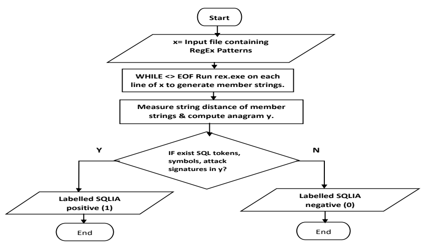

Start
x= Input file containing RegEx Patterns
WHILE <> EOF Run rex.exe on each line of x to generate member strings.
Measure string distance of member strings & compute anagram y.
IF exist SQL tokens, symbols, attack signatures in y?
Y
N
Labelled SQLIA positive (1)
Labelled SQLIA negative (0)
End
End

Fig. 4. Data set feature values labelling Flow Chart.

IV. APPROACH: PREDICTIVE ANALYTICS EXPERIMENT AND DEPLOYMENT

Predictive Analytics provides a robust approach to big data mining. We apply predictive analytics in this paper in mitigating SQLIA. The approach is built on MAML studio, which is a Cloud-based machine learning platform. The experimental steps are detailed below.

A. A high-level overview of the experimental steps:

1) Data set extraction: The learning data containing feature values is used here in the MAML studio to train a supervised learning model contains pattern-driven data set described in detail in Section III (B). We obtained *479,000* member strings with additional *862* unique SQL tokens extracted from Microsoft SQL reserved keywords [28]. The feature values labelling is described above in Section III (B4) which is represented by binary values of *0* or *1*.

2) Text preprocessing: This stage involves R Scripting that incorporates all the defined RegEx constraints detailed in Section III (B) to parsed learning data. The feature values are parsed for patterns, duplicates, normalised to lower cases and the removal of the missing attribute values which results in the pruning of the feature values to *362,603*. The data set is sampled to provide an even distribution of the feature values. The imbalanced feature values of majority labelled SQLIA negatives over positives were corrected with Synthetic Minority Over-Sampling Technique (SMOTE)[17]. The entire data of *725206* is split equally with *362,603* row items labelled SQLIA negatives and *362,603* row items as SQLIA positives. The SMOTE improves the accuracy and F1 score statistical measures in an evenly distributed corpus.

3) Features hashing to the matrix: The hashing is to transform the data set feature values into a binary vector matrix of 2^{15} *(32,768)* columns required for training a classifier in ML by setting the hashing bit size to *1* (unigrams of N-grams) where *N = 1*. The hashing procedure creates a dimensional input of the matrix with a faster lookup of feature weights by substituting the string comparison with the hash value comparison. Applying hashing to text features improves the performance and scalability of predictive analytics that is lacking in existing SQLIA signature-based detection approaches. We use a unigram hashing of strings into the binary matrix as the intention is to analyse intercepted strings as a unit at the proxy. We observed analysing a string as a unigram offers a better prediction of TP and TN than analysing a phrase of a group of strings together (N-Grams > 1).

4) Split of vector matrix between training and testing data: We divide the matrix values of the hashed features into a ratio of *80:20 (training: test)* of which *80%* forms the training data input to the classifier while *20%* as test data for evaluations. We further optimised the classifier in the MAML studio with Tune Model Hyperparameters (TMH) module to improve TP and TN predictions, but with low FN results of *162* achieved in TC SVM as shown in Section V Table 1V.

5) Training the prediction model: Both TC LR and TC SVM algorithms employ linear kernel which offers a binary prediction at the proxy a linear separation between SQLIA positive and negative presence in a web request. This linearity of the classes which can be demarcated in a straight line makes algorithms (classifiers) using linear kernel a preferred choice in binary classification. Also, the two classifiers show good accuracy and fast training times in performance metrics. TC SVM has the advantage of being scalable with significant features set as used here in vectorization (hashing) that generates higher dimensionality of hashed columns. TC SVM and TC LR algorithms are trained with the training data of the partitioned matrix values. We achieved in the trained model AUC values of *0.984* and *0.986* for TC LR and TC SVM respectively in ROC curve but with TC SVM having fewer FP and FN.

B. Publishing and consuming the prediction web service

The system requirements regarding RAM and the hard disk are very low as the one-off workload of training the classifier including retraining is handled in the Cloud by the MAML platform. The solution is scalable, and it is meant to detect and prevent SQLIA in web requests as illustrated in Fig. 5 below detailing how the excellently trained model is consumed in a web proxy and web form in an ongoing SQLIA detection and prevention. Critical to the deployment in every new web application domain context, the administrator or system expert needs to feed the data engineering or text preprocessing module with new rules that match the patterns present in the new expected input data which triggers the retraining of the classifier to adapt to a new application domain context [19].

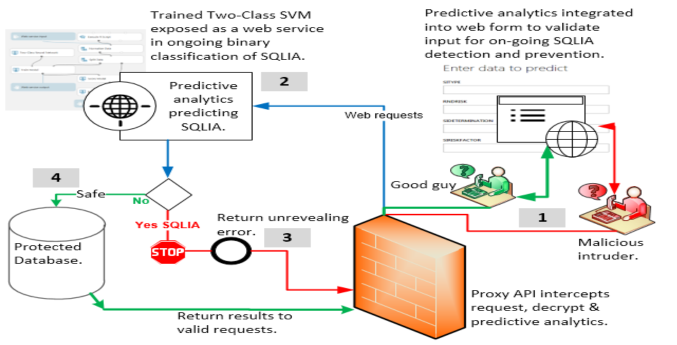

Trained Two-Class SVM exposed as a web service in ongoing binary classification of SQLIA.
Predictive analytics integrated into web form to validate input for on-going SQLIA detection and prevention.
Enter data to predict
Predictive analytics predicting SQLIA.
2
Web requests
4
Safe
No
Yes SQLIA
STOP
Good guy
1
Malicious intruder.
Return unrevealing error.
3
Protected Database.
Proxy API intercepts request, decrypt & predictive analytics.
Return results to valid requests.

Fig. 5. A design overview including consuming a trained model at the web proxy API and client forms in an ongoing SQLIA detection and prevention [19].

V. EVALUATION AND PERFORMANCE METRICS

ROC curve and AUC are widely used by data scientists to measure the performance metrics in ML analytics. It provides a valid empirical statistical measure for the evaluation of results. ROC curve, which has its origin in World War II to predict radar images for threats has been widely accepted in interpreting medical test results [29] and recently in computing data sciences.

We extracted an evenly distributed data set of *725206* attribute values or row items by preprocessing and equal balancing of feature values as described in Section IV (A2). These feature values to be predicted contain an equal representation of labelled strings deemed valid web requests and SQLIA threat as described in data extraction labelling in Section III (B4). The string attributes values were hashed to obtain a matrix represented by X_{ij} that refers to the element in rows i and columns j of the input variables matrix X. The output variable to predict Y is a single vector representation by Y_i where i is the index or row count. Thus, the learning data l is represented by Equation (1) below as a dimension of matrix X_{ij} to predict an output vector matrix Y_i where n is the top row count of index i. A function of X denoted as $f(X)$ to predict a labelled output Y. Therefore, $f(X)=Y$, where x is input and y is the predicted output.

$$l = (x_j^{(i)}, y^{(i)} \ldots\ldots x_j^{(i)}, y^{(i)}) = X_{ij}, Y_i \ldots\ldots X_{nj}, Y_n \qquad (1)$$

We split the hashed string features matrix and associated predictor variables into a ratio *80:20* % of 725206 with *580164* matrix values as a training set while the remaining *145042* as a test set. We observed a split ratio of *80:20* (training: test) resulted in better performance metrics compared with other split ratios that were tried. The training set is evaluated under various binary classification algorithms or classifiers to select a better performing classifier determined by the AUC performance value. The linear kernel driven algorithms presented here are implemented in MAML (Azure ML) Studio using the TC LR and TC SVM classifiers. We observed linear kernel driven classifiers are better performing in the binary classification of two classes in predicting the discrete value of 0/1. The AUC provides an overall performance measure between the classifier algorithms as illustrated in Fig. 6 for which the TC SVM with AUC value of *0.986* was observed to be better performing than the TC LR of AUC value 0.984.

While the *80%* training sample is the part fed to the classifier to train the prediction model, the remaining *20%* is the test data vectors, values unseen by the classifier which is the subject of this empirical evaluation presented here. The test data input variables of *145042* rows are scored to generate score probabilities of a range $0 \leq x \leq 1$ where x is the input matrix to predict output y. The score probabilities provide a measure of observations that are correctly predicted as TP and TN including the two prediction errors of FP and FN within the range $\{0,1\}$. These prediction observations of TP, TN, FP and FN are presented in tables below, are calculated to determine how many of these observations score probability values fall within each score bins set of $\{0,1\}$.

Therefore, the expected output is $y = 0$ or SQLIA negative if the function of the predictor variables x is closer to 0 expressed as $f(x) \approx 0$. Conversely, the prediction output is $y = 1$ or SQLIA positive when the function of predictor variables x is closer to 1 denoted as $f(x) \approx 1$. Also, the prediction errors rate is used to gauge the performance of a classifier as illustrated in Table IV (a snippet of Table V and VI) with TC SVM achieving low FN (*162*). However, *1923* FP events were observed in TC SVM which indicates such web requests that are falsely alarmed will be referred to the system monitor for a further review.

TABLE IV. A SNIPPET OF PREDICTION OBSERVATIONS AT DEFAULT THRESHOLD AT 0.5 REPEATED ACROSS {0,1}

Events	Positive	Negative
Positive	TP TC SVM=72359, LR = 69421	FP TC SVM = 1923, LR = 2088
Negative	FN TC SVM= 162, LR = 3100	TN TC SVM=70598, LR = 70433

The MAML studio sets by default the cut-off threshold for the prediction of TP and TN including the errors of FP and FN to be *0.5*. This cut-off of *0.5* is a predetermined threshold employed by classification algorithms; it is a trade-off between the cost function of x to predict y against the performance metrics statistical measures; which the latter is the default. Therefore $f(x) < 0.5$ score probability value is predicted as SQLIA negative *(0)* while $f(x) \geq 0.5$ is predicted as SQLIA positive *(1)*. The *145042* values of score probabilities are partitioned into ten score bins of *0.1* increments of the set $\{0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$. The predicted observations are aggregated across these score bins using the score probability values as shown in Table V and VI.

TABLE V. OBSERVATIONS AT VARIOUS CUT-OFF POINTS BETWEEN *{0,1}* OF THE TC LR TRAINED MODEL

Score Bins	TP	FN	FP	TN	PE	NE	PO	NO	FPR	TPR/Recall	Accuracy	P	FI Score
1	0	72521	0	72521	72521	72521	0	145042	0	0	0.5	0	0
0.9	52479	20042	1399	71122	72521	72521	53878	91164	0.01929096	0.72363867	0.85217385	0.974034	0.830370493
0.8	58756	13765	1568	70953	72521	72521	60324	84718	0.02162132	0.81019291	0.89428579	0.974007	0.884579773
0.7	62924	9597	1674	70847	72521	72521	64598	80444	0.02308297	0.86766592	0.92229147	0.974086	0.917801326
0.6	66303	6218	1760	70761	72521	72521	68063	76979	0.02426883	0.91425932	0.94499524	0.974142	0.94325101
0.5	69421	3100	2088	70433	72521	72521	71509	73533	0.02879166	0.95725376	0.96423105	0.970801	0.963979726
0.4	71704	817	4794	67727	72521	72521	76498	68544	0.06610499	0.9887343	0.96131465	0.937332	0.962347083
0.3	72325	196	8374	64147	72521	72521	80699	64343	0.11547	0.99729733	0.94091367	0.896232	0.944067354
0.2	72371	150	12508	60013	72521	72521	84879	60163	0.17247418	0.99793163	0.91272873	0.852637	0.919580686
0.1	72390	131	18708	53813	72521	72521	91098	53944	0.25796666	0.99819363	0.87011348	0.794639	0.88486056
0	72521	0	72521	0	72521	72521	145042	0	1	1	0.5	0.5	0.666666667
Abbreviations & Formula	True Positive (TP)	Faslse Negative (FN)	False Positive (FP)	True Negative (TN)	Positive Event (PE) =TP+FN	Negative Event (NE) =FP+TN	Positive Observations(PO) =TP+FP	Negative Observations (NO) =FN+TN	False Positive Rate (FPR) =FP / (FP+TN)	True Positive Rate (TPR) =TP / PE	(TP+TN) / Total events (TE) TE = 145042	Precision (P) =TP / PO	FI Score =2*(TPR*P) / (TPR+P)

TABLE VI. OBSERVATIONS AT VARIOUS CUT-OFF POINTS BETWEEN *{0,1}* OF THE TC SVM TRAINED MODEL

Score Bins	TP	FN	FP	TN	PE	NE	PO	NO	TE	FPR	TPR/Recall	Accuracy	P	FI Score
1	0	72521	0	72521	72521	72521	0	145042	145042	0	0	0.5	0	0
0.9	56446	16075	1505	71016	72521	72521	53878	91164	145042	0.02075261	0.77834007	0.878794	0.97403	0.86525845
0.8	63346	9175	1692	70829	72521	72521	60324	84718	145042	0.023331173	0.87348492	0.925077	0.973984	0.92100117
0.7	67886	4635	1806	70715	72521	72521	64598	80444	145042	0.024903132	0.93608748	0.955592	0.974086	0.95470878
0.6	71088	1433	1887	70634	72521	72521	68063	76979	145042	0.026020049	0.98024021	0.97711	0.974142	0.9771815
0.5	72359	162	1923	70598	72521	72521	71509	73533	145042	0.026516457	0.99776616	0.985625	0.974112	0.98579729
0.4	72324	197	1921	70600	72521	72521	76498	68544	145042	0.026488879	0.99728355	0.985397	0.974126	0.98556886
0.3	72340	181	2179	70342	72521	72521	80699	64343	145042	0.030046469	0.99750417	0.983729	0.970759	0.98394995
0.2	72355	166	6175	66346	72521	72521	84879	60163	145042	0.08514775	0.99771101	0.956282	0.921368	0.9580208
0.1	72376	145	13108	59413	72521	72521	91098	53944	145042	0.180747646	0.99800058	0.908626	0.846661	0.91612291
0	72521	0	72521	0	72521	72521	145042	0	145042	1	1	0.5	0.5	0.66666667

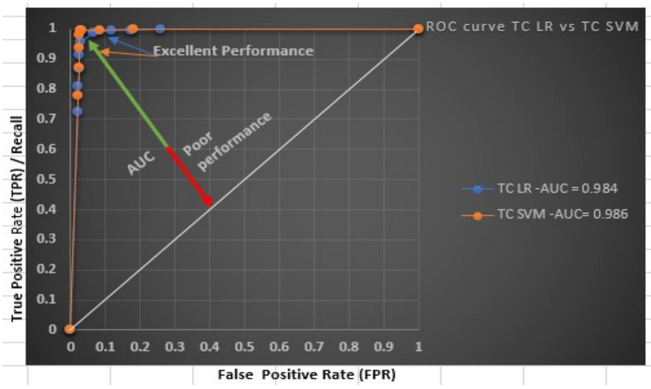


Fig. 6. ROC curve of the trained models comparing the performance in AUC of TC LR against TC SVM classifiers.

The statistical measures provide the performance metrics of a trained model. We calculated the statistical measures at the various thresholds *{0,1}* as shown in Table V where TC LR at default has the following: Accuracy = *0.964*, Precision = *0.971*, Recall = *0.957*and F1 Score = *0.964.* Table VI is calculated as the preceeding, where TC SVM has an improved performance metrics with Accuracy = *0.986*, Precision = *0.974*, Recall = *0.998*, F1 Score = *0.986* and AUC of 0.986 as shown in Fig. 6.

VI. CONCLUSION AND FUTURE WORK

We demonstrated in this paper a pattern-driven data set generated using SFA in the absence of a pre-existing data set to apply predictive analytics to SQLIA detection and prevention in a big data context. We empirically evaluated our results in ROC curve. Future work involves employing multi-class classifier in predicting the different SQLIA types.

REFERENCES

[1] CodeCurmudgeon, "SQLi Hall-of-Shame," *The Code Curmudgeon*, 2016. [Online]. Available: http://codecurmudgeon.com/wp/sql-injection-hall-of-shame/. [Accessed: 12-Aug-2016].

[2] M. Veanes, N. Bjorner, and L. de Moura, "Symbolic Automata Constraint Solving," vol. 6397. 2010.

[3] M. Veanes, "Rex @ rise4fun from Microsoft," *Microsoft Research*. [Online]. Available: http://rise4fun.com/rex.

[4] E. Lawrence, "Fiddler free web debugging proxy," *Telerik*. [Online]. Available: http://www.telerik.com/fiddler. [Accessed: 11-Feb-2015].

[5] G. Wassermann and Z. Su, "An analysis framework for security in Web applications," *SAVCBS 2004 Specif. Verif. Component-Based Syst.*, p. 70, 2004.

[6] I. Medeiros, N. Neves, and M. Correia, "Detecting and Removing Web Application Vulnerabilities with Static Analysis and Data Mining," *IEEE Trans. Reliab.*, vol. 65, no. 1, pp. 54–69, Mar. 2016.

[7] S. Thomas, L. Williams, and T. Xie, "On automated prepared statement generation to remove SQL injection vulnerabilities," *Inf. Softw. Technol.*, vol. 51, no. 3, pp. 589–598, 2009.

[8] A. Owasp, "OWASP Top 10 Proactive Controls 2016," 2016.

[9] W. G. J. Halfond and A. Orso, "AMNESIA: Analysis and Monitoring for NEutralizing SQL-injection Attacks," *Proc. 20th IEEE/ACM Int. Conf. Autom. Softw. Eng.*, pp. 174–183, 2005.

[10] Y. S. Jang and J. Y. Choi, "Detecting SQL injection attacks using query result size," *Comput. Secur.*, vol. 44, pp. 104–118, 2014.

[11] W. G. J. Halfond, A. Orso, and P. Manolios, "WASP: Protecting web applications using positive tainting and syntax-aware evaluation," *IEEE Trans. Softw. Eng.*, vol. 34, no. 1, pp. 65–81, 2008.

[12] A. Kie, P. J. Guo, and M. D. Ernst, "Automatic Creation of SQL Injection and Cross-Site Scripting Attacks," 2009.

[13] C. Bockermann, M. Apel, and M. Meier, "Learning SQL for database intrusion detection using context-sensitive modelling (extended abstract)," in *Lecture Notes in Computer Science*, 2009, vol. 5587 LNCS, pp. 196–205.

[14] R. Komiya, I. Paik, and M. Hisada, "Classification of malicious web code by machine learning," in *Proceedings of 2011 3rd International Conference on Awareness Science and Technology, iCAST 2011*, 2011, pp. 406–411.

[15] J. Choi, C. Choi, H. Kim, and P. Kim, "Efficient malicious code detection using N-gram analysis and SVM," in *Proceedings - 2011 International Conference on Network-Based Information Systems, NBiS 2011*, 2011, pp. 618–621.

[16] Y. Wang and Z. Li, "SQL Injection Detection via Program Tracing and Machine Learning," *LNCS*, vol. 7646, pp. 264–274, 2012.

[17] C. I. Pinzón, J. F. De Paz, Á. Herrero, E. Corchado, J. Bajo, J. M. Corchado, C. I. Pinz??n, J. F. De Paz, ??lvaro Herrero, E. Corchado, J. Bajo, and J. M. Corchado, "IdMAS-SQL: Intrusion Detection Based on MAS to Detect and Block SQL injection through data mining," *Inf. Sci. (Ny).*, vol. 231, pp. 15–31, 2013.

[18] M.-Y. Y. Kim and D. H. Lee, "Data-mining based SQL injection attack detection using internal query trees," *Expert Syst. Appl.*, vol. 41, no. 11, pp. 5416–5430, 2014.

[19] S. O. Uwagbole, W. J. Buchanan, and L. Fan, "Applied Machine Learning Predictive Analytics to SQL Injection Attack Detection and Prevention," in *3rd IEEE/IFIP Workshop on Security for Emerging Distributed Network Technologies (DISSECT)*, 2017.

[20] S. O. Uwagbole, W. Buchanan, and L. Fan, "Applied web traffic analysis for numerical encoding of SQL injection attack features," in *European Conference on Information Warfare and Security, ECCWS*, 2016, vol. 2016–Janua.

[21] S. Uwagbole, W. Buchanan, and L. Fan, "Numerical Encoding to Tame SQL Injection Attacks," in *IEEE/IFIP DISSECT*, 2016.

[22] A. Liu, Y. Yuan, D. Wijesekera, and A. Stavrou, "SQLProb : A Proxy-based Architecture towards Preventing SQL Injection Attacks," *Conf. Proc. 2009 ACM Symp. Appl. Comput.*, pp. 1–8, 2009.

[23] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, 1943.

[24] M. O. Rabin and D. Scott, "Finite Automata and Their Decision Problems," *IBM J. Res. Dev.*, vol. 3, no. 2, pp. 114–125, 1959.

[25] M. Veanes, P. de Halleux, N. Tillmann, P. De Halleux, and N. Tillmann, "Rex: Symbolic regular expression explorer," in *ICST 2010 - 3rd International Conference on Software Testing, Verification and Validation*, 2010, pp. 498–507.

[26] M. Veanes, "Applications of symbolic finite automata," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2013, vol. 7982 LNCS, pp. 16–23.

[27] M. van der Loo, "{stringdist}: an {R} Package for Approximate String Matching," *R J.*, vol. 6, no. 1, pp. 111–122, 2014.

[28] Microsoft, "Reserved Keywords (Transact-SQL)," *MSDN*. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms189822.aspx.

[29] T. G. Tape, "Using the Receiver Operating Characteristic (ROC) curve to analyze a classification model," *Univ. Nebraska Med. Cent.*, pp. 1–3, 2000.