Applied Web Traffic Analysis for Numerical Encoding of SQL Injection Attack Features

Solomon Ogbomon Uwagbole, William J. Buchanan, Lu Fan
Edinburgh Napier University, Edinburgh, United Kingdom
s.uwagbole@napier.ac.uk
b.buchanan@napier.ac.uk
l.fan@napier.ac.uk

Abstract: SQL Injection Attack (SQLIA) remains a technique used by a computer network intruder to pilfer an organisation's confidential data. This is done by an intruder re-crafting web form's input and query strings used in web requests with malicious intent to compromise the security of an organisation's confidential data stored at the back-end database. The database is the most valuable data source, and thus, intruders are unrelenting in constantly evolving new techniques to bypass the signature's solutions currently provided in Web Application Firewalls (WAF) to mitigate SQLIA.
There is therefore a need for an automated scalable methodology in the pre-processing of SQLIA features fit for a supervised learning model. However, obtaining a ready-made scalable dataset that is feature engineered with numerical attributes dataset items to train Artificial Neural Network (ANN) and Machine Leaning (ML) models is a known issue in applying artificial intelligence to effectively address ever evolving novel SQLIA signatures.
This proposed approach applies numerical attributes encoding ontology to encode features (both legitimate web requests and SQLIA) to numerical data items as to extract scalable dataset for input to a supervised learning model in moving towards a ML SQLIA detection and prevention model. In numerical attributes encoding of features, the proposed model explores a hybrid of static and dynamic pattern matching by implementing a Non-Deterministic Finite Automaton (NFA). This combined with proxy and SQL parser Application Programming Interface (API) to intercept and parse web requests in transition to the back-end database. In developing a solution to address SQLIA, this model allows processed web requests at the proxy deemed to contain injected query string to be excluded from reaching the target back-end database.
This paper is intended for evaluating the performance metrics of a dataset obtained by numerical encoding of features ontology in Microsoft Azure Machine Learning (MAML) studio using Two-Class Support Vector Machines (TCSVM) binary classifier. This methodology then forms the subject of the empirical evaluation.

Keywords: SQL injection, SQLIA, numerical encoding, input neurons, Azure Machine Learning, Training Data

1. Introduction

Continuous innovations in internet applications have seen an astronomical growth of data with trending research areas of big data and the Internet of Things (IoT). Mining of large data to address security vulnerabilities are beyond the limitation of signature approach but an alternative machine learning approach provides a solution. Over the years, there are ongoing issues in securing web driven applications. This is evidently demonstrated by the continuous and intrusive attacks originating from many hacking groups including (but not limited to) governments, lone wolf (Khandelwal, 2015), Anonymous and LulzSec rogue groups (Schwartz, 2011; Schone et al., 2014). A hacker, or an intruder, is an individual or group that breaches the security of a computer system by employing an array of techniques to disrupt and pilfer confidential data. A typical method used to steal confidential data is by SQLIA which often leaves any signature driven Web Application Firewalls (WAF) (Appelt et al., 2015) playing catch-up every time there are new attack signatures. Machine learning approaches are able to effectively protect against novel signatures by classifying new attack signatures not trained for as unknown, thereby dropping or referring such requests at the interim.

Current SQLIA research areas are lacking in ready-made robust dataset samples with numeric encoded features. The few non-standard sample files that exist would normally contain unprocessed strings of repeating features of the variations that exist within SQLIA types. The scheme presented here provides a technique with just a regular expression to numerically encode features from any file containing SQLIA features. It also includes a full implementation on how you would extract the dataset from real-time web traffic and deployment to MAML studio to train a supervised learning model implementing TCSVM (Microsoft Azure, 2016).
The proposed model is built on MAML (Microsoft Azure, n.d.). The methodology includes: extraction of

dataset attributes items and labelling; classification of SQLIA features and validation of the supervised learning model (Kotsiantis, 2007). The model is then exposed as a web service in ongoing SQLIA detection and prevention.

Though this proposed model simultaneously delivers a self-contained SQLIA detection and prevention solution, it is intended for evaluating the fitness of dataset extracted by numerical encoding of features ontology using TCSVM statistical model binary classifier. This is the subject of the empirical evaluation in Section 5.

The paper is laid out in six sections ending with a conclusion and future work. Section 2 covers background and theory (attack intent, injection mechanism and SQL Types); Section 3 is focused on related work; with Sections 4 and 5 detailing the features encoding and results.

2. Background Theory

The proposed model implements a traditional NFA (Rabin & Scott, 1959; MSDN, n.d.) to match patterns of injection mechanisms and SQLIA types which are then encoded to numerical dataset items. Dataset are input for ANN and ML and this contains numerical attributes of independent x-variables(predictor) and y-dependent variable (what to predict or labelled attribute). Legitimate web requests are expected valid requests that a monitored web application will generate. There are RegEx patterns that validate SQL parsed assembled web requests in-transition at the proxy. The use of proxy and SQL parser in this model means that during backhaul, web traffic can be laid bare and analysed for the attack intent, injection mechanism and SQLIA type.

Vulnerable web applications can be susceptible to the following injection mechanisms: injection through web forms; cookies; operating system server instrumentations; and Trojan horse used in second-order attack. The seven notable SQLIA types include: Tautology; Invalid/Logical Incorrect; Union; Piggy-backed; Store procedure; Time-based; and Alternate encoding obfuscation (Halfond et al., 2008).

These notable SQLIA types have derivations within a SQLIA type which can best be described as an intruder tweaking a known attack signature to evade detection e.g. a SQL injection tautological attack of 1=1 can also be written as 1<1, ‘a’=’a’ etc. to achieve the same attack. It is these derivations that exist within a SQLIA type that account for numerous SQLIA features in any large sample dataset populated with repeated strings. This paper proposes a scheme to account for these derivations with the random risk attribute to account for the fact that there are many derivations within a SQLIA type. These random decimal values provide a way to derive large dataset items of both legitimate web requests and SQLIA features (Uwagbole et al., 2016).

An intruder would normally first establish a technique to use in SQLIA by probing the target website with a series of trial runs to determine the hotspot and method that best suits the intended target. It may also deploy a combination of SQLIA types for a successful attack. As an example a normal web request will have a query string of http://bsid/bsid/Data Page.aspx?LoginName=bob&Password=@bob which is evaluated in SQL parser to SELECT loginName, password FROM tblUser WHERE loginName='bob' AND password ='@bob'. This legitimate web request can be injected to any of the SQLIA types discussed below.

The scheme presented here for encoding numerical attributes from features of strings as detailed in Section 4 is fully replicable with basic background experience in .NET C# and R scripting using open source software of fiddler proxy (Lawrence, n.d.), SQL Script Dom Parser API (MSDN, n.d.), RegEx (MSDN, n.d.) and MAML studio.

Throughout this section simple queries examples (not exhaustive) are presented in the tables to illustrate the different SQLIA types with red flag alerts being matched in the encoding of features into numerical data. The section below provides a high-level overview of the SQLIA types which forms the bases for the pattern matching of both legitimate web requests and SQLIA payload that are encoded into numerical data. The layout of the tables below has: attack intent; web request query string; parsed transact Structured Query Language (tSQL) and an indication (pattern) of red flags being pattern matched.

2.1 Tautology

This SQL injection type of attack is carried out by injecting vulnerable sites with query strings that are altered in transition to retrieve data from the database. The classical example is by assigning altered strings to the WHERE clause after incorrectly terminating the query with a single quote (‘) or with a tautology statement like *OR 1=1* as shown in Table 1*.* The outcome is always parsed by SQL parser to be true meaning that the security validation of login credentials will be bypassed to retrieve all the records at the affected back-end database table.

Table 1: Tautology queries

Attack intent	Injectable hotspots, circumventing authentication, extracting data	Pattern
Query string	http://localhost/bsid/DataPage.aspx?LoginName=bob'OR%201=1--&Password=	`--,1=1,1<1 ‘a’=’a’ etc.
Parsed tSQL	SELECT loginName, password FROM tblUser WHERE loginName= 'bob' OR 1=1--	

2.2 Invalid/Logical Incorrect Query

This is often used to probe the vulnerabilities of the target prior to an attack. The information gained during an initial attack is useful to the intruder to determine what form of further attack to carry out on the target. The example query string in Table 2 will return a divide by zero error if the login account is not *sa* or using *dbo* schema.

Table 2: Logical incorrect queries

Attack intent	Detecting vulnerabilities, extracting data, database finger-printing	Pattern
Query string	http://localhost/bsid/DataPage.aspx?LoginName=';IF((SELECT%20user)%20=%20'sa'%20OR%20(SELECT%20user)%20=%20'dbo')%20SELECT%201%20ELSE%20SELECT%201/0;--2%80%99&Password=	Duplicate SELECT IF, ELSE sa, dbo,1/0
Parsed tSQL	SELECT loginName, password FROM tblUser WHERE loginName=’; IF ((SELECT user) = 'sa' OR (SELECT user) = 'dbo') SELECT 1 ELSE SELECT 1/0; --	

2.3 Union

This attack exploits the UNION command ability to query multiple database tables to retrieve confidential data far beyond the tables used in the web application as shown in Table 3. There are intermediate steps of using ORDER BY to get column names (sqlinjection, 2016).

Table 3: Union queries

Attack intent	*Data extraction and bypassing authentication*	Pattern
Query string	http://localhost/bsid/DataPage.aspx?LoginName=&Password=UNION%20SELECT%20CreditNo,%20CustAddress%20from%20tblCreditinfo	‘ UNION SELECT
Parsed tSQL	*SELECT loginName, password FROM tblUser WHERE loginName=' ' UNION SELECT CreditNo, CustAddress from tblCreditinfo*	

2.4 Piggy-backed

The intruder exploits the semicolon (;) to append a valid SQL statement to further SQLIA. In Table 4, an intruder uses another SQLIA type (sqlinjection, 2016) to obtain table name and then the SQL statement is piggy-backed to run SELECT * from tblCreditinfo in order to gain unauthorised credit card information.

Table 4: Piggy-backed queries

Attack intent	Extracting data beyond the scope of the web application and executing commands	Pattern
Query string	http://localhost/bsid/DataPage.aspx?LoginName=;%20SELECT%20*%20from%20tblCreditInfo%20--	‘ ;
Parsed tSQL	SELECT loginName, password FROM tblUser WHERE LoginName=''; SELECT * from CreditCardInfo --	

2.5 Store procedure

The intruder elicits database information by exploiting xp_cmdshell if enabled to trigger a Trojan horse file for malicious attack. Also stored procedures are vulnerable to privilege escalation, buffer overflows, and even manipulated to gain elevated permission access to perform operating system wide operations (Halfond et al., 2008). In the example query presented in Table 5, an attacker runs xp-cmdshell against a spurious text files loaded in the target to circumvent the security in the database.

Table 5: Store procedure queries

Attack intent	Privilege escalation, remote command, performing denial of service	Pattern
Query string	http//bsid/bsid/login.aspx? LoginName=’ ’;exec xp_cmdshell 'attrib "c:\test\spuriousfile.vbs" +r'	;, Exec, xp_cmdshell Attrib c:\test\ spuriousfile.vbs " +r
Parsed tSQL	SELECT * FROM tblUser where loginname ='‘; exec xp_cmdshell 'attrib "c:\test\ spuriousfile.vbs " +r'	

2.6 Time-based

This is a timed delayed type of SQLIA where an intruder probes a site to elicit a response after a period of time. The query shown in Table 6 will display a response after ten seconds that provides an intruder with information to further more attacks.

Table 6: Time based queries

Attack intent	Probing of injectable hotspots and database schema	Pattern
Query string	http://localhost/bsid/DataPage.aspx?LoginName=';%20waitfor%20delay%20'00:00:10'--&Password=	‘ waitfor delay '00:00:10' --
Parsed tSQL	SELECT loginName, password FROM tblUser WHERE loginName=' '; waitfor delay '00:00:10'--	

2.7 Alternate encoding obfuscation

A combination of escaped-encoding and Unicode character which the computer systems interpret as normal without distinction from intended obfuscation; an intruder can circumvent solutions being provided in pattern matching of SQLIA as it becomes unreadable as shown in Table 7.

***Table 7**: Alternate encoding obfuscation*

Attack intent	Obfuscating data to evade detection	Pattern
Query string	http%3A%2F%2Flocalhost%2Fbsid%2FDataPage.aspx%3F%0ALoginName%3Dbob%27OR%25201%3D1--26Password%3D%0A	% Hex values Numeric values --
Parsed tSQL	SELECT loginName, password FROM tblUser WHERE loginName=' '; waitfor delay '00:00:10'--	

Not limited to the above injection mechanisms and SQLIA types which are the subject of numerical encoding of features in this paper; there are other forms of pattern evading techniques like the use of comments, whitespace, character casing and encryption that are exploited by an intruder in SQLIA. Section 3 discusses related work.

3. Related work

Whilst it is acknowledged that there are existing foundational works (Boyd & Keromytis, 2004; Gould et al., 2004; Buehrer et al., 2005; Halfond & Orso, 2005) that share similarities with the approach used, these similarities do not extend to the proposed scheme to encode numerical attributes as presented in this paper. In recent work (Uwagbole et al., 2016) introducing the numerical encoding of features ontology to obtain dataset attributes was applied to ANN and statistical ML models implemented using Two-Class Averaged Perceptron (TCAP) and Two-Class Logistic Regression (TCLR) classifier respectively that gave a prediction rate of Area Under Curve (AUC) of 0.914 (91.4%). In this paper, it presents a full implementation on how you would extract the dataset from real-time web traffic directly from MAML and trained a supervised learning model implementing TCSVM classifier with an improved performance results of AUC of 0.944 (94.4%) shown in confusion matrix (performance metrics) in Section 5.

Most recent works are derivations of foundational works on SQL injection that often benchmark detection rates against these earlier research foundational works (Buehrer et al., 2005; Wu et al., 2015), (Halfond & Orso, 2005; Wang et al., 2015) devoid of today's big data challenges with growing internet data. As intruders become ever smarter in developing novel techniques to tweak known SQLIA types with whitespaces, character casing, comments, encryption and encoding, so it becomes evident that the static signature methods (Boyd & Keromytis, 2004; Kar et al., 2015) lack the ability to cope, having to continuously create new signatures. Exploring an alternative machine learning approach is a direction towards a better prediction of scalable processing demand of growing internet data traffic.

SQLProb (Liu et al., 2009) uses proxy in their approach, as does this proposed model, but this proposed model goes further in applying supervised learning in prediction of true positives and negatives as against genetic algorithm. SQL parsing tree (Buehrer et al., 2005) and CANDID (Bisht et al., 2010) are centred on code analysis algorithms for detection of SQLIA. The benefit asserted by the authors of the approach is that it can be retrofitted, but as these methods lack pattern matching the approach will not be functional in today's high volumes of web traffic.

Though AMNESIA (Halfond & Orso, 2005) is a hybrid of static and dynamic approach that uses NFA as does this paper; in the AMNESIA experiment, a Java based NFA implementation of SQLIA pattern searches were used solely in SQLIA detection and prevention but in the approach presented here, the matched pattern is numerically encoded into dataset items that are fed into ANN and ML implemented on MAML platform.

SQLrand (Boyd & Keromytis, 2004) is another related work that employs proxy and parser. Although the authors claim the methodology to have a negligible impact on performance, it is unlikely that it would be scalable in today's large data driven web traffic due to their approach being geared towards signatures as against pattern matching employed in AMNESIA. JDBC Checker (Gould et al., 2004) is a static approach, which although it explores finite state automaton that is also used in this model, it lacks proxy to backhaul web traffic as to lay bare web requests for thorough analysis.

4. Features encoding

Section 4.1 discusses features encoding to numerical attributes items destined as input dataset to a supervised learning model implementing TCSVM while Section 4.2 is a high level overview of the full implementation steps on MAML studio.

4.1 Numerical attributes extraction from features steps

The dataset input to the supervised learning model are features numerically encoded from legitimate web requests, injection mechanisms and known SQLIA types. The attributes (predictors) are scaled down in this paper for simplicity, but the approach can be replicated for as many attributes that are desired and attributed to SQLIA behaviour patterns. Table 8 shows the semantics of how the numeric attributes data items are abstracted.

Table 9 contains numerical attributes data items that are extracted as detailed in Table 8 which has the dataset items of both numerical encoded features of normal and SQLIA:

- Sitypes *p* of recognised patterns p *{w_0… w_n}*
- Sidetermination *v* cross validated feature types v *{w_0… w_n}* using a method of NFA backtracking (MSDN, n.d.)
- Rndrisk *r* is randomised values to account for the variations within a feature *r_0… r_n.*

Siriskfactor *l* is the likeliness of a feature being a risk factor which can either be -1 likeliness or 1 for remote likeliness. This is collated from the predictor independent variables (x-attributes) of *p, d, r*. The dependent y-variable or what to predict (commonly known as a labelled class) is inferred from *l* which could be a possible 1 for SQLIA or 0 for normal as shown in Table 8 with steps detailing how these values were extrapolated. The scheme can be replicated with any pattern or text processing tool like RegEx to assign number range (numeric attributes data items). However, a sum of the attributes needs to be less than or greater than the value set for the threshold in likeliness of it being normal or suspect.

***Table 8**: Numerical encoding of features algorithm* (Uwagbole et al., 2016)

```
1.  Extracting the primary independent predictors
    Get matched patterns (p) of SQLIA and legitimate payload using NFA(RegEx)
        if feature matched a static pattern
        assign a numeric value from 1 to 9
    Validate (v) matched pattern(p)
        randomly assign values 0.01 to 0.09
    Randomization (r) to account for the variations within injection mechanisms and SQL types
        randomly assigned a decimal value less than 0.01
        computed against web requests total count
2.  Calculating the likeliness of being normal (n) threshold
        Sum p + v + r and take the minimal from the set of values above 9
        If   n >= 9
            Then
                Normal = 1
        Else
                Suspect = -1
3.  Calculating y –variables or what to predict
        If normal(n)
            Then
                0
        Else
                1
```

***Table 9**:* Encoded numerical attributes data items for input ML

Sitype(p)	sidetermination (v)	Rndrisk (r)	Siriskfactor (l)	Siclass
p{w_0 }	*v{w_0 }*	*r_0*	*l_0*	*y_0*
1	*0.04*	*0.389420*	*-1*	*1*
9	*0.09*	*0.142830*	*1*	*0*
p{w_n }	*v{w_n }*	*r_n*	*l_n*	*y_n*

4.2 The implementation steps on MAML studio

Figure 1 is the MAML experiment screen capture that is carried out for the proposed model. The steps include: streaming the extracted numerical dataset items into MAML studio; classification of SQLIA features; validation of supervised learning model which is then exposed as web services in ongoing detection and prevention. Below are the nine key steps:

1. The web traffic is backhauled at proxy to analyse web requests for injection mechanisms, legitimate web requests and SQLIA features.
2. The web requests payload in-transition to the back-end database are assembled to full queries and parsed by a SQL parser API at the proxy.
3. The payload is subjected to pattern matching employing a traditional NFA approach (RegEx) while at the same time numerically encoding both matched and unknown patterns in web requests.
4. The numerical encoded dataset is streamed into the MAML studio by R scripting call to traditional NFA implemented using Microsoft RegEx patterns matching at the proxy.
5. This is followed by scrubbing of the missing data and column casting as the input dataset is feature engineered with missing values to achieve a better confusion matrix that can accurately predict SQLIA. This helps remove over-fitting from the binary classification.
6. Next step is logistic normalisation which is the rescaling of the numeric data as to constrain the values (Microsoft Azure, 2015b) for a good fitting of the numeric data into the statistical trained model.

7. Next is the splitting of data between training and testing data. A repeated training of the statistical model with different splits ratios found using 80% training data to 20 % test data that gave the highest prediction rates shown in Section 5 confusion matrix (figure 2).
8. The supervised learning model was trained using TCSVM classifier algorithm. It was scored and evaluated to establish how well the prediction of the supervised learning classification model is performing. This gave AUC value of 0.944 (94.4%) shown in confusion matrix in Section 5 (figure 2).
9. Finally, a predictive web service was generated from the trained supervised learning model built using MAML studio to obtain an API code to integrate into web form for ongoing SQLIA prediction.

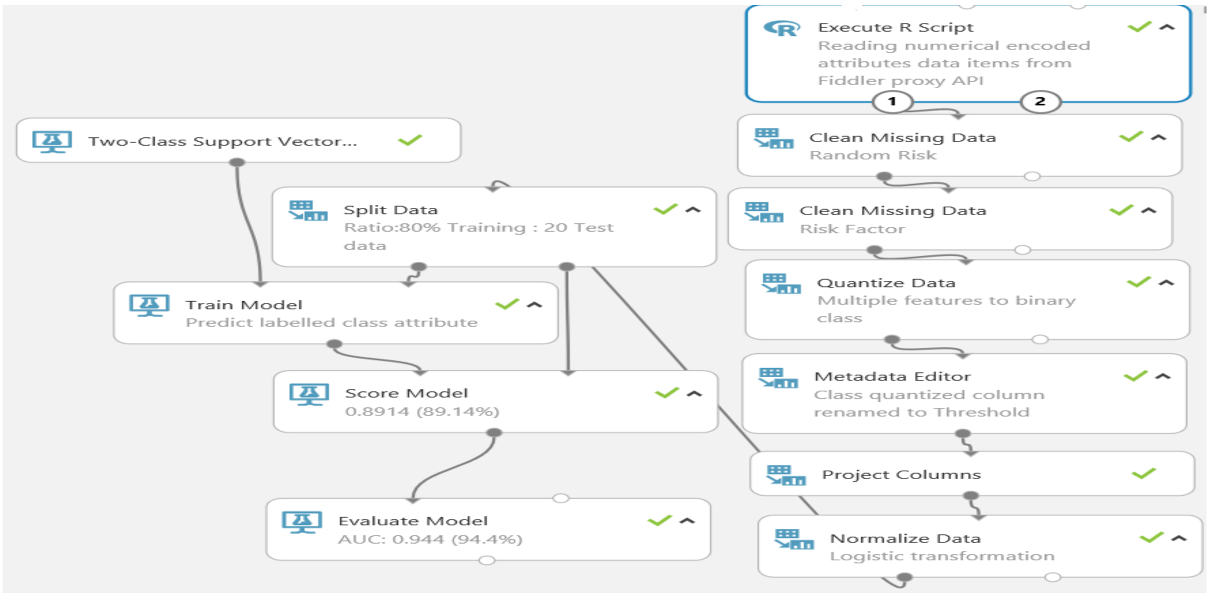

Execute R Script
Reading numerical encoded attributes data items from Fiddler proxy API
1
2
Two-Class Support Vector...
Clean Missing Data
Random Risk
Split Data
Ratio:80% Training : 20 Test data
Clean Missing Data
Risk Factor
Train Model
Predict labelled class attribute
Quantize Data
Multiple features to binary class
Score Model
0.8914 (89.14%)
Metadata Editor
Class quantized column renamed to Threshold
Project Columns
Evaluate Model
AUC: 0.944 (94.4%)
Normalize Data
Logistic transformation

Figure 1: The MAML studio predictive experiment layout.

Whilst all new web requests from new IP addresses go through the pattern matching process but the established authenticated IP addresses are validated through the web service of the supervised classification trained model for ongoing scalable SQLIA detection and prevention.

5. Evaluation and Result

There were four attributes or x-variables (sitype, sidetermination, rndrisk, siriskfactor) with one labelled attribute or y-variable (siclass). ML and ANN approaches need a large dataset numerically encoded from behaviour or patterns of normal and SQLIA features to accurately predict. The numerical attributes encoding ontology also detailed in this paper provides a way to generate as many desired dataset attributes and rows items. There were 59702 rows of attributes data items of which 80% (47762 rows items) were used as training data and a further 20% (11940 rows items) were used as the test data. This test data is the subject of computational statistics in any performance metrics (confusion matrix). There are 1:8 ratios between normal and attack features in the distribution of the attributes data items (rows) and a further outliers of data items with missing values. Through repeated training of the supervised classifier using different normalisation and split ratios, an optimum classification was achieved at normalisation using logistic data transformation method with a split data ratio of 80:20 between training to test data which achieved an AUC value of 0.944 (94.4%) on TCSVM classification model.

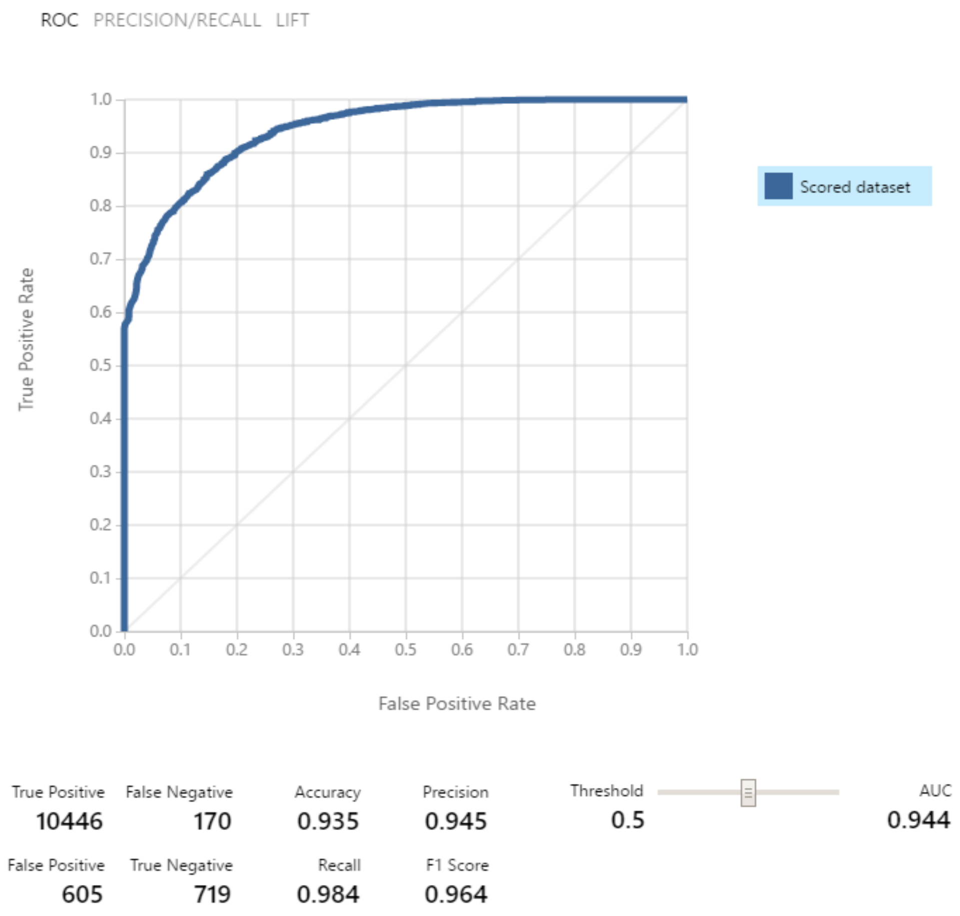
The performance metrics or confusion matrix (accuracy, precision, recall and F1 score) from Figure 2 are calculated as follows:

- Accuracy is the proportion of actual true results to the total cases that is calculated as:
  true positives + true negatives (10446 + 719) / total cases (11940) = 0.935.
- Precision is the proportion of true overall positive results returned by the model that is calculated as:
  true positives (10446) / true positives + false positives (10446+605) = 0.945.

- Recall is the true positive rate which is fraction of total correct results returned by the model calculated as: true positives (10446) / all positive cases (true positives (10446) + true negatives (170) = 0.984.
- F1 score is a measure of accuracy that balances precision and recall (Microsoft Azure, 2015a) calculated as: 2 *(recall (0.984)*precision(0.945))/(recall(0.984)+ precision(0.945)) = 0.964.

Receiver Operating Characteristic (ROC) is a graphical plot (de Ruiter, 2015) in Figure 2 using variation in threshold discrimination to illustrate the performance of the binary classifier system implementing TCSVM classifier with a curve towards the upper left corner indicating a better performing model. Area Under the Curve (AUC) or area below the curve in the graph plot is a measure of true positives on the y-axis against false positives on the x-axis. An excellent prediction model is inferred with AUC 0.944 as shown in Figure 2 which indicates the scheme presented will efficiently predict true positives and negatives as required in any effective ML SQLIA detection and prevention model.

Figure 2: ROC plot and confusion matrix of evaluation results



True Positive	False Negative	Accuracy	Precision	Threshold	AUC
10446	170	0.935	0.945	0.5	0.944
False Positive	**True Negative**	**Recall**	**F1 Score**		
605	719	0.984	0.964		

6. Conclusion and future work

The work presented in this paper demonstrates the fitness of numerical encoding of web requests primed as dataset (detail in Table 8 & 9) to a statistical binary classifier implemented in MAML using TCSVM classifier. The evaluation results presented above in the ROC graph plot and confusion matrix empirically evaluates the proposed scheme to apply ML in real-time for a scalable prediction of SQLIA. This approach is geared towards leveraging recent advancement in the field of artificial intelligence to build a scalable application that can predict SQLIA in web requests with a high degree of accuracy in true positives and negatives.

Whilst an excellent supervised predicting model has been achieved and tested with a further coding to map back the numerical features to interpret string features but a future work is needed to directly interpret the SQLIA strings from the trained supervised learning model.

References

Appelt, D., Nguyen, C.D. & Briand, L. (2015) Behind an application firewall, are we safe from SQL injection attacks? In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation, ICST 2015 - Proceedings*.

Bisht, P., Madhusudan, P. & Venkatakrishnan, V.N. (2010) CANDID: Dynamic candidate evaluations for automatic prevention of SQL injection attacks. *ACM Trans. Inf. Syst. Secur.*, 13(2), pp.14:1–14:39. Available at: http://doi.acm.org/10.1145/1698750.1698754.

Boyd, S.W. & Keromytis, A.D. (2004) SQLrand: Preventing SQL injection attacks. In *Applied Cryptography and Network Security*. pp. 292–302. Available at: http://link.springer.com/chapter/10.1007/978-3-540-24852-1_21.

Buehrer, G.T., Weide, B.W. & Sivilotti, P.A.G. (2005) Using Parse Tree Validation to Prevent SQL Injection Attacks. In *Proceedings of the 5th international workshop on Software engineering and middleware SEM 05*. p. 106. Available at: http://portal.acm.org/citation.cfm?doid=1108473.1108496.

Gould, C., Su, Z. & Devanbu, P. (2004) JDBC checker: a static analysis tool for SQL/JDBC applications. In *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*. pp. 697–698.

Halfond, W.G.J. & Orso, A. (2005) AMNESIA: Analysis and Monitoring for NEutralizing SQL-injection Attacks. *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, pp.174–183. Available at: http://doi.acm.org/10.1145/1101908.1101935.

Halfond, W.G.J., Viegas, J. & Orso, A. (2008) A Classification of SQL Injection Attacks and Countermeasures. *Preventing Sql Code Injection By Combining Static and Runtime Analysis*, p.53.

Kar, D., Panigrahi, S. & Sundararajan, S. (2015) Sqlidds: SQL injection detection using query transformation and document similarity. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. pp. 377–390. Available at: http://www.scopus.com/inward/record.url?eid=2-s2.0-84922376264&partnerID=tZOtx3y1.

Khandelwal, S. (2015) Fourth, a 16-year-old Hacker, Arrested over TalkTalk Hack. *The Hacker News*. Available at: http://thehackernews.com/2015/11/talktalk-hacker.html [Accessed January 23, 2016].

Kotsiantis, S.B. (2007) Supervised Machine Learning : A Review of Classification Techniques. *Informatica*, 31, pp.249–268. Available at: http://books.google.com/books?hl=pt-BR&lr=&id=vLiTXDHr_sYC&pgis=1.

Lawrence, E. Fiddler free web debugging proxy. *Telerik*. Available at: http://www.telerik.com/fiddler [Accessed February 11, 2015].

Liu, A. et al. (2009) SQLProb : A Proxy-based Architecture towards Preventing SQL Injection Attacks. *System*, pp.2054–2061. Available at: http://portal.acm.org/citation.cfm?id=1529282.1529737.

Microsoft Azure (2015a) Machine Learning / Evaluate. *MSDN Library*. Available at: http://tinyurl.com/zybaw94 [Accessed February 1, 2016].

Microsoft Azure Microsoft Azure Machine Learning Studio. *Microsoft Azure Machine Learning*. Available at: https://studio.azureml.net/ [Accessed January 25, 2015].

Microsoft Azure (2015b) Normalize Data. *MSDN Library*. Available at: https://msdn.microsoft.com/en-us/library/azure/dn905838.aspx [Accessed January 6, 2016].

Microsoft Azure (2016) Two-Class Support Vector Machine. *MSDN Library*. Available at: https://msdn.microsoft.com/en-us/library/azure/dn905835.aspx [Accessed January 22, 2016].

MSDN Matching Behavior. *MSDN Library*. Available at: https://msdn.microsoft.com/en-us/library/0yzc2yb0(v=vs.100).aspx [Accessed February 3, 2016a].

MSDN Microsoft.SqlServer.TransactSql.ScriptDom Namespace. Available at: https://msdn.microsoft.com/en-us/library/microsoft.sqlserver.transactsql.scriptdom.aspx [Accessed October 25, 2015b].

Rabin, M.O. & Scott, D. (1959) Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, 3(2), pp.114–125.

de Ruiter, A. (2015) Using ROC plots and the AUC measure in Azure ML | Andreas De Ruiter's BI blog. *MSDN*. Available at: https://blogs.msdn.microsoft.com/andreasderuiter/2015/02/09/using-roc-plots-and-the-auc-measure-in-azure-ml/ [Accessed March 15, 2016].

Schone, M. et al. (2014) War on Anonymous: British Spies Attacked Hackers, Snowden Docs Show. *Nbcnews.Com*. Available at: http://www.nbcnews.com/news/investigations/war-anonymous-british-spies-attacked-hackers-snowden-docs-show-n21361.

Schwartz, M.J. (2011) Sony Hacked Again, 1 Million Passwords Exposed. *InformationWeek*. Available at: http://www.darkreading.com/attacks-and-breaches/sony-hacked-again-1-million-passwords-exposed/d/d-id/1098113?

sqlinjection (2016) SQL Injection Using UNION. *sqlinjection*. Available at: http://www.sqlinjection.net/union/

[Accessed February 3, 2016].

Uwagbole, S., Buchanan, W. & Fan, L. (2016) Numerical Encoding to Tame SQL Injection Attacks. In *IEEE/IFIP DISSECT*. In press.

Wang, Y. et al. (2015) Detecting SQL Vulnerability Attack Based on the Dynamic and Static Analysis Technology. In *2015 IEEE 39th Annual Computer Software and Applications Conference*. IEEE, pp. 604–607. Available at: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=7273432 [Accessed March 20, 2016].

Wu, T.. et al. (2015) Towards SQL injection attacks detection mechanism using parse. In *In Genetic and Evolutionary Computing*. Springer International Publishing, pp. 371–380.