Applied Machine Learning Predictive Analytics to SQL Injection Attack Detection and Prevention

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*— The back-end database is pivotal to the storage of the massive size of big data Internet exchanges stemming from cloud-hosted web applications to Internet of Things (IoT) smart devices. Structured Query Language (SQL) Injection Attack (SQLIA) remains an intruder's exploit of choice on vulnerable web applications to pilfer confidential data from the database with potentially damaging consequences. The existing solutions of mostly signature approaches were all before the recent challenges of big data mining and at such lacks the functionality and ability to cope with new signatures concealed in web requests. An alternative Machine Learning (ML) predictive analytics provides a functional and scalable mining to big data in detection and prevention of SQLIA. Unfortunately, lack of availability of readymade robust corpus or data set with patterns and historical data items to train a classifier are issues well known in SQLIA research. In this paper, we explore the generation of data set containing extraction from known attack patterns including SQL tokens and symbols present at injection points. Also, as a test case, we build a web application that expects dictionary word list as vector variables to demonstrate massive quantities of learning data. The data set is pre-processed, labelled and feature hashing for supervised learning. The trained classifier to be deployed as a web service that is consumed in a custom dot NET application implementing a web proxy Application Programming Interface (API) to intercept and accurately predict SQLIA in web requests thereby preventing malicious web requests from reaching the protected back-end database. This paper demonstrates a full proof of concept implementation of an ML predictive analytics and deployment of resultant web service that accurately predicts and prevents SQLIA with empirical evaluations presented in Confusion Matrix (CM) and Receiver Operating Curve (ROC).**

Keywords*— *SQLIA; SQLIA analytics; SQL Injection; SQLIA big data; SQLIA hashing

I. INTRODUCTION

Research over the years has mostly pinpointed developers' lack of security awareness in web development to sanitised input as the cause of SQLIA, and at such have gravitated towards code based sanitation for their proposed solutions to address SQLIA. Also, SQLIA vulnerability is a sequel to a design fallout of the well-intentioned free text processing of the SQL engine itself, and as a consequence both legacy and cloud deployments lacking sanitation becomes vulnerable. A search of SQL hall of shame [1] which reports the recent trends in data pilfering by SQLIA shows the prevalence of this form of attack and so the ability to secure back-end database from SQLIA in an era of big data remains a topical issue.

The SQL language syntax closely resembles plain English [2], and the SQLIA keywords are also in plain text. Therefore, the SQLIA problem in a big data context is a plausible candidate for predictive analytics of a supervised learning model trained via both known historical attack signatures and safe web requests patterns.

The attack signatures at injection points will contain patterns of SQL tokens and symbols as SQLIA positive while valid web requests would take the form of expected data from the application. In this paper, we build a predictive analytics web application with quantities of learning data to train a classifier. The learning data are labelled vector matrix, or features of both patterns of dictionary word list (SQLIA negative) and SQL tokens (SQLIA positive).

The contributions this paper makes provide a representative data set that undergo feature hashing to train a supervised learning model implementing Support Vector Machine (SVM) algorithm that accurately predicts SQLIA thereby preventing malicious web requests from reaching the target back-end database. It also offers a context of SQLIA detection and prevention in big data internet.

Also, this paper presents a proof of concept of a working prototype using ML algorithms of Two-Class Support Vector Machine (TCSVM) implemented on Microsoft Azure Machine Learning (MAML) [3] to predict SQLIA. This methodology then forms the subject of the empirical evaluation in Receiver Operating Curve (ROC).

This paper is laid out in six sections ending with a conclusion and future work summary. Section II details related articles and Section III focuses on background; with Sections IV and V detailing predictive analytics experiment, evaluation and results.

II. RELATED WORK

Though there are many previous proposals to detect and prevent SQLIA, few mitigate SQLIA by applying SVM machine learning. The few that do employ SVM are lacking in data engineering (text pre-processing). Also, to date, none have discussed applying ML to predicting SQLIA in a context of big data focusing on patterns and text pre-processing on MAML.

We suggest patterns exist in any data input to a web application to generate training data, and also by applying text pre-processing to such training data improves the prediction

accuracy of the resultant trained model. The below discusses a few related works not limited to ML and proxy filters that are related to this paper, but including some popular approaches.

A. *Support Vector Machine Learning:*

Applying ML requires robust data set items with patterns to train a classifier implementing SVM algorithm to predict SQLIA accurately. Unfortunately, as there is no standardised data set, researchers have presented various approaches for extracting data sets with most proposals suffering from limited data engineering. Bockermann et al. [4] propose using tree kernels for analysing SQL statement in addition to exploring feature vectorisation of data input to an SVM classifier but found there to be drawbacks in the tree-kernels computational overhead. Choi et al. [5] trains an SVM classifier using feature vectorisation by N-Grams but would need various patterns to improve the accuracy of the approach. Kar et al. [6] propose using SQL queries graph of tokens and centrality of nodes to train an SVM classifier but suffers from complexities. There have been previous approaches of numerical encoding of synthetic training data of SQLIA patterns for training a classifier to simulate SQLIA prediction of any size [7], [8].

B. *Proxy Filters:*

This method intercepts web requests at a proxy for SQLIA detection and prevention having the advantage of being able to decrypt obfuscated internet traffic for thorough analysis. Buehrer et al. [9] propose a SQL parsing tree which uses a combination of proxy and SQL parser tree for SQL syntax sequence alignment. The model proposed in this paper uses proxy API to backhaul web requests for predictive analytics of incoming web requests for SQLIA negatives and positives.

C. *Others*

White-box testing is a static code analysis penetration testing to detect error and correctness. Gould et al.[10] developed a tool named JDBC Checker for code analysis to only detect some SQLIA types but not to prevent. Wassermann and Su [11] extended white box testing to detect tautology.

Black-box testing is a runtime dynamic penetration testing to detect error and correctness. Apelt [12] proposed a machine learning tool to automate penetration testing of Web Application Firewall (WAF) for vulnerabilities which look plausible, but the tool relies heavily on synthetic attack features lacking in reality of new attack signatures.

Hybrid of a static and dynamic approach employs pattern matching between valid request against dynamic web requests to detect and prevent SQLIA as applied in Halfond & Orso [13] widely referenced AMNESIA tool to mitigate SQLIA. However, these are approaches that scaled well in traditional string matching at the time and are not functional in big data scenarios that require predictive analytics techniques.

III. BACKGROUND THEORY

The approach presented in this paper intercepts web requests of any intent at the proxy and applies ML predictive analytics to the requests at injection point to predict SQLIA.

A web proxy API is the most suitable to intercept requests originating from any injection mechanisms. Injection mechanisms can originate from any: Web page forms e.g. login screen; second-order injection by concealing a Trojan horse for the attack at a later date; exploiting web-enabled server variables to gain access to the back-end database; and, through cookies that have stored state information used to obtain unauthorised access to the back-end database.

SQLIA types are techniques an intruder would employ at injection points in any combination to carry out an attack that includes: Tautology; Invalid/Logical Incorrect; Union; Piggy-backed; Store procedure; Time-based; and, Alternate encoding obfuscation. SQLIA types provide an extract for the SQLIA positive in data set items during labelling. Further reading on SQLIA types in the paper [14].

A. *SQL Language Structure*

The constituents of a SQL element are tokens pivotal to SQLIA negative labelling in this article. SQL tokens comprise of keywords, identifiers, operators, literals and punctuation symbols. The SQL statement is subdivided into the following primary language elements: Clauses (UPDATE, SET, WHERE, etc.); predicates (e.g. loginName = ‘bob’); and, expressions (as in ‘bob’ OR 1=1) shown in Fig. 1.

An intruder will normally exploit the expression part (injection spot) of the predicate after the WHERE clause used to control the results of data requested from the database including updating the database. A tautological SQLIA type (e.g., ‘a’=’a’ or 1=1) maliciously embedded in expression to return all the data beyond the valid scope defined by the developer, likewise a full SQL query inserted into the SQL element’s expression spot. In applying predictive analytics, we analysed the predicate and expression for SQLIA signatures.

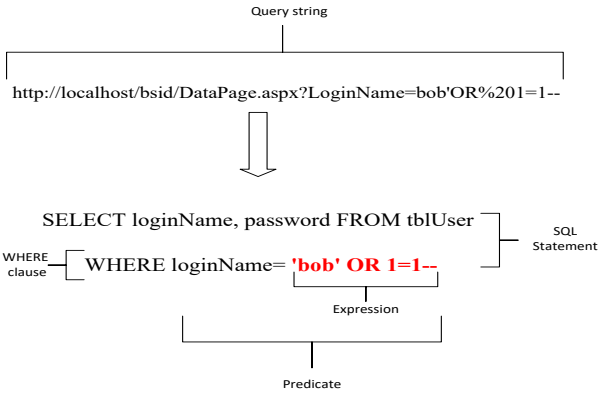

Fig. 1. Query string and SQL query element.

B. *Injection Point*

The WHERE clause determines the level of access to the back-end database from the front-end web application. In reality, this could be validating login credentials, filtering or setting criteria of the data to return, or inserting a new/ updating an existing record. The injection point is the SQL element predicate and expression after the WHERE clause in SQL query.

The SQL element's expression injection point is the location for the predictive analytics for SQLIA detection and prevention presented here. This spot has also been explored in a Proxy-based Architecture towards Preventing SQL Injection Attacks (SQLProb) by Liu et al. [15].

C. SQLIA Corpus and Labelling

We tested our approach on a web application expecting dictionary word list as a valid input. The corpus comprises of the derivation of dictionary word list (labelled SQLIA negative), SQL tokens and symbols (labelled SQLIA positive if present at injection points). The labelling procedure is detailed in Fig. 2. When this approach is applied to any domain, data set can be generated based on the pattern of data input expected. Also, the existing studies on SQLIA types provide patterns to generate malicious requests. Various approaches have existed on extracting sample data set with most researchers resorting to network tools to produce sample web requests [6], [12] which often results in mere duplication of the same string but lack patterns implementation (regular expression pattern matching on the data) for improving the performance of ML models.

```
FOR each row of the data set items
        IF matched pattern of lowercase of data items has no:
                SQL tokens
                SQL symbols
                disjointed text
                single quotes
                semi-colons
                comments
                whitespaces
                special characters
                hex/obfuscated values
            Return 0 for SQLIA negative in labelling
        ELSE
            Return 1 for SQLIA positive in labelling
END FOR
```

Fig. 2. Data set features labelling procedure.

IV. Predictive Analytics Experiment and Deployment

Predictive analytics provides a scalable and functional approach to big data mining in mitigating SQLIA vulnerabilities.

A. Experimental steps

Below gives a high-level overview of the steps:

1) Data set extraction: This research uses a combination of data set of extracted dictionary word list of 479,000, words in addition to 862 unique SQL tokens extracted from Microsoft SQL reserved keywords website [16]. The data set items are labelled based on the exhibition of SQLIA types characteristics which are: the presence of SQL tokens in injection point; disjointed text; single quotes; semicolons; comments; hex; etc. The data set items labelling are represented in binary values of *0* (SQL negative) or *1* (SQL negative) shown in Fig. 2 above.

2) Text pre-processing: This stage involves R Scripting and regular expression pattern matching applying the procedure described in Fig. 2 integrated to train a model for ongoing detection and prevention. In a real domain application, the data set items are expanded with patterns of both valid requests and bad requests. There were *362,603*-row items after text pre-processing of parsing data set for: patterns, duplicates, normalised to lower cases and the removal of the missing words. The data set is sampled as to provide an even distribution of row items (records). The imbalanced data set (majority negatives over positives) were corrected with Synthetic Minority Over-Sampling Technique (SMOTE) [17] to have a data set items of *725206* split equally, *362,603*-row items of attack/respondent (positives) and *362,603*-row items of non-attack/non-respondent (negatives). These actions improve both the trained model recall and precision.

3) Feature hashing: Machine learning takes in numerical input as vectors. In this scheme, we set the hashing bits to *15* and *N*-grams to *1* (unigram) to analyse every single item present at the injection point. The process of feature hashing allows us to translate the data set text items into a binary vector matrix of 2^{15} *(32,768)* columns suitable for training a model in ML. The hashing procedure creates a dimensional input matrix or vectors that make a lookup of feature weights faster by augmenting the string comparison with hash value comparison. Applying hashing to text features improves performance and scalability in big data predictive analytics lacking in existing SQLIA signature based detection.

4) Filter-based feature selection for top relevant vectors: Creating a dimension to accommodate the size of data by selecting next hashing bits that fit the data set can sometimes generate too much dimension and sparse data which are reduced by filter based featured selection. In this experiment, we used filtered based selection to have reduced computation complexity without affecting the prediction accuracy in classification. The Chi-squared scored function is used to rank the top 5000 hashing features in descending order to return the most appropriate labels to improve SQLIA prediction accuracy.

5) Split of vectors between training and testing data: We divide the vectors of hashed features into different ratios of which samples of 80% were used for the training while 20% as test data for evaluations. The vector split at this ratio provided an excellent CM in the assessment of the trained classifier.

6) Train prediction model: TCSVM classifier is used to predict binary labelled outcomes whether SQLIA is negative or positive in a web request. The SVM algorithm is provided with data set items input of the labelled class of what is being predicted to train the model to an excellent performance to accurately predict SQLIA in a proxy intercepted web requests.

B. Publishing and consuming the prediction web service

The system requirements regarding RAM and the hard disk is 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. 3.

The trained model exposed as a web service. The web service is called in a custom built dot NET application for this research named NETSQLIA for an ongoing SQLIA detection and prevention. Critical to the deployment in every new domain, the administrator or system expert need to feed the data engineering or text pre-processing module with a new rule that matches the patterns present in the new data set which

triggers the retraining of the classifier to adapt to a new environment.

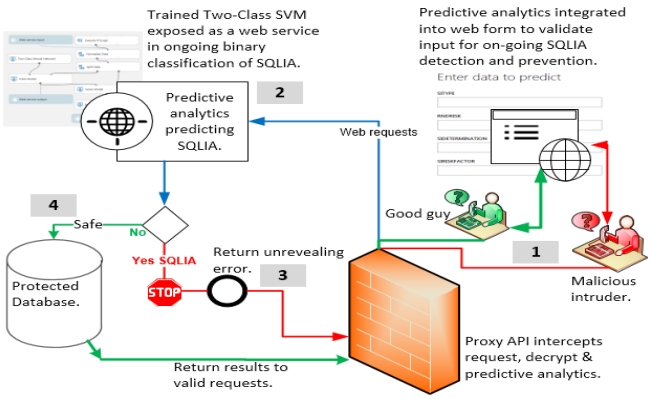

Fig. 3. A custom application is consuming the trained SVM web service for ongoing SQLIA detection and prevention.

V. EVALUATION AND PERFORMANCE METRICS

Table 1 is an excellent CM (accuracy: *0.986*, precision: *0.974*, recall: *0.997* and F1 Score: *0.985*) at a threshold of *0.5*, but this calculation is repeated across the score bins/ thresholds between *0.0* and *1* for the ROC graph values in Fig. 4. The ROC is plotted with a ratio of False Positives Rate (FPR) on the x-axis (specificity) against recall or True Positive Rate (TPR) on the y-axis (sensitivity). On the x-axis, a higher value implies bad performance, while on the y- axis a higher value of *0.986* in Area Under Curve (AUC) achieved here at 0.5 score bin indicates an excellent performance in SQLIA prediction.

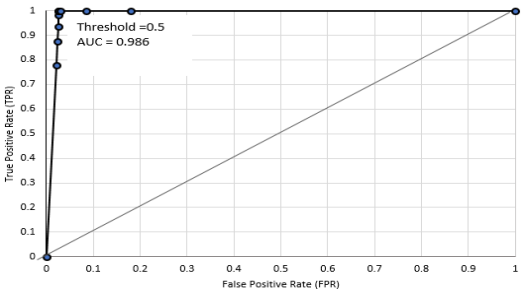

Fig. 4. ROC plot of FPR (x-axis) against TPR(y-axis).

TABLE I. CONFUSION MATRIX AT THRESHOLD 0.5

Terminology	Formula	Values	Performance metrics	
True Positive (TP)	-	72359	***Accuracy(A)= (TP+TN)/TE***	0.986
False Negative (TN)	-	162		
False Positive (FP)	-	1923		
True Negative (FN)	-	70598	***Precision (P)=TP/(PO)***	0.974
Positive events (PE)	***TP+FN***	72521		
Negative events (NE)	***FP+TN***	72521	***Recall (R)= TP/PE***	0.997
+ observations (PO)	***TP+FP***	74282		
- observations (NO)	***FN+TN***	70760		
Total events (TE)	***PO+ NO***	145042	***F1Score=2* (R*P)/(R+P)***	0.985

VI. CONCLUSION AND FUTURE WORK

We demonstrated in this paper applied predictive analytics to SQLIA detection and prevention in big data context with an excellent result that is empirically evaluated in the confusion matrix and the ROC graph presented above. In benchmarking this paper against existing works, the methodology proposed here is functional in a big data context which is lacking in existing works before now on SQLIA to our knowledge. Future work involves employing multi-class classifier to identify and group the different SQLIA types as they are predicted.

REFERENCES

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

[2] Microsoft, "Access SQL: basic concepts, vocabulary, and syntax - Access," *MS Office*, 2007. [Online]. Available: https://support .office.com/en-gb/article/Access-SQL-basic-concepts-vocabulary-and-syntax-444d0303-cde1-424e-9a74-e8dc3e460671.

[3] Microsoft Azure, "Two-Class Support Vector Machine," *MSDN Library*,2016.[Online]. Available: https://msdn.microsoft.com/ library/azure/12d8479b-74b4-4e67-b8de-d32867380e20/?f=255 &MSPPError=-2147217396. [Accessed: 22-Jan-2016].

[4] 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 (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5587 LNCS, pp. 196–205.

[5] 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.

[6] D. Kar, S. Panigrahi, and S. Sundararajan, "SQLiGoT: Detecting SQL Injection Attacks using Graph of Tokens and SVM," *Comput. Secur.*, vol. 60, pp. 206–225, 2016.

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

[8] 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.

[9] G. T. Buehrer, B. W. Weide, and P. A. G. Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks," in *Proceedings of the 5th international workshop on Software engineering and middleware SEM 05*, 2005, no. September, p. 106.

[10] C. Gould, Z. Su, and P. Devanbu, "JDBC checker: a static analysis tool for SQL/JDBC applications," in *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, 2004, pp. 697–698.

[11] G. Wassermann, C. Gould, Z. Su, and P. Devanbu, "Static checking of dynamically generated queries in database applications," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 4, p. 14, Sep. 2007.

[12] D. Appelt, "Automated Security Testing of Web-Based Systems Against SQL Injection Attacks," 2016.

[13] 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.

[14] W. G. J. Halfond, A. Orso, D. A. Kindy, and A. S. K. Pathan, "AMNESIA: Analysis and Monitoring for NEutralizing SQL-injection Attacks," *Int. J. Commun. Networks Inf. Secur.*, vol. 5, pp. 80–92, 2013.

[15] A. Liu, Y. Yuan, D. Wijesekera, and A. Stavrou, "SQLProb : A Proxy-based Architecture towards Preventing SQL Injection Attacks," *System*, pp. 2054–2061, 2009.

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

[17] Nitesh V. Chawla et. al, "SMOTE," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.