ISSN: 1803-3814 (Printed), 2571-3701 (Online)
https://doi.org/10.13164/mendel.2019.1.023

MACHINE LEARNING BLUNTS THE NEEDLE OF ADVANCED SQL INJECTIONS

Marina Volkova✉**, Petr Chmelar, Lukas Sobotka**
GREYCORTEX, Brno, Czech Republic
marina.volkova@greycortex.com✉, petr.chmelar@greycortex.com, lukas.sobotka@greycortex.com

Abstract
SQL injection is one of the most popular and serious information security threats. By exploiting database vulnerabilities, attackers may get access to sensitive data or enable compromised computers to conduct further network attacks. Our research is focused on applying machine learning approaches for identification of injection characteristics in the HTTP query string. We compare results from Rule-based Intrusion Detection System, Support Vector Machines, Multilayer Perceptron, Neural Network with Dropout layers, and Deep Sequential Models (Long Short-Term Memory, and Gated Recurrent Units) using multiple string analysis, bag-of-word techniques, and word embedding for query string vectorization. Results proved benefits of applying machine learning approach for detection malicious pattern in HTTP query string.

Keywords: *SQL injection identification, machine learning, deep learning, recurrent neural networks, text analysis, web application firewall, intrusion detection system.*

Received: 29 April 2019
Accepted: 31 May 2019
Published: 24 June 2019

1 Introduction

Web application vulnerabilities are among the highest severity cybersecurity problems. Structured Query Language Injection Attacks (SQLIA) have been rated as the number-one attack in web application threats according to Open Web Application Security Project (OWASP) [1]. The objective of these attacks is to gain access to the database. Exploiting vulnerabilities in web applications allows attackers to bypass the authentication scheme, get access to sensitive data, make changes in database schema, use the compromised server, or attack other computers inside the network [2].

The essence of the injection is to add a malicious query to the legitime database request. If a user‘s input is passed unvalidated and unsanitized as a part of an SQL query, the attacker can manipulate the query itself, and force it to return different data than the query was supposed to return.

Generally, web application firewalls (WAF) and intrusion detection systems (IDS) are used to protect web applications from database-related attacks. These tools match the rules and signatures from their knowledge base with the current query. But it can be problematic to identify unknown attacks and obfuscated requests. Moreover, the complexity of detection for SQLIA by WAF and IDS comes because attacks are performed through ports used for regular web traffic (open in firewalls) and work at the application level [3].

Applying an analytical approach to identify SQLIA from the query allows improved detection tools. Text analysis techniques assume specific characters, symbols or keywords, and their context to determine threats. Currently, machine learning algorithms are successfully used in cybersecurity applications [4].

The goal of this research is to analyze existing IDS and Machine Learning (ML) approaches to SQL injection identification and to find the best technique to apply in Network Traffic Analysis, in “real world” settings.

The paper is structured as follows. The next section of the paper is dedicated to a general description of web application logic, SQLIA principles and types, and it explains why these attacks are still successful. In the third section of the article, we consider some advantages and disadvantages of existing models. The fourth section contains a description of our experiments, including the experiment scheme, data description, feature extraction techniques, and classification algorithms. The experimental results and practical applications are discussed in the fifth section. Our research is summarized in the last section.

2 Background and Motivation

Let us start with understanding web application logic. Suppose we have a web application consisting of three layers: user interface (thin client - browser), application server, and database server (see Fig. 1) [5].

1. The user fills the login form and sends an HTTP request to the application server. For example, a login request: login=’User1’ and password=’12345’.

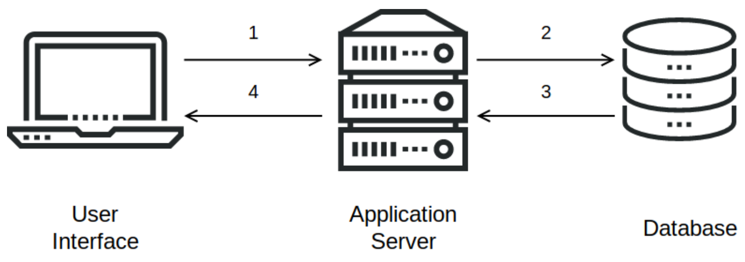


Figure 1: Web application processing

2. The server transforms user information to a dynamic SQL request:
   SELECT * FROM account WHERE login='User1' AND password='12345'.
3. If there is a positive answer from the database server, the user data from the 'accounts' table is sent to the application server.
4. The application server sends the reply to the user.

To explain how SQL injection attacks (SQLIA) work [6]: The attacker inserts a custom-created request from the user side: login=user1' OR 1=1 –. The web application server transforms it, and the database server receives the request: SELECT * FROM account WHERE login='User1' OR 1=1 – AND password='anypass'. Here a tautology command "OR 1=1 –" is used. It means all inputs are true and upcoming conditions will be executed. So attacker gets access to the database server and can run any command.

2.1 SQLIA Types

Several group of SQLIA are distinguished [7]. The first type are error-based attacks. The goal is to get additional information about the database structure and information scheme. This type can be extended with tautology and comments. Tautology means adding conditional statements that always return TRUE to a valid query, so that the query may return all records from the table. Using the comment line causes the database to ignore a part of a query.

The next type is UNION based injection. The attacker integrates a malicious query with a valid request tricking the application into returning data from a table different from the one that was intended by the developer. The last big group of injections are blind attacks; used to identify injectable parameters. Blind attacks can be attributed boolean and time-base attacks. During boolean blind attacks the information is gained from the server by asking true/false questions. The "true" response keeps the application operating normally. The "false" response raises an error message with meaningful database information (e.g. type of database) or makes the web application operate significantly differently from its regular functionality. A timing attack is based on gaining database information by observing timing delays in the response from the database [7], [8].

There are many approaches and solutions for determining and preventing an attack targeting databases. Most detection methods, like Web Applications Firewall (WAF), use attack type characteristics and corresponding signatures [9]. Vulnerabilities in these tools are described in detail in OWASP reports [10]. For example, the request /?id=1+union+select+1,2,3/* will be blocked by WAF signatures. However, attackers use plenty variances in request obfuscation (Table 1).

Table 1: Web Application Firewall Bypassing

Bypassing	Example
Normalization	index.php?id=1/*uni X on*/union/*sel X ect*/select+1,2,3/*
Pollution	/?id=1/**/union/*,*/select/*,*/pwd/*,*/from/*,*/users
Fragmentation	/?id=1+union/*&b=*/select+1,pass/*&c=*/from+users–
Blind Injection	/?id=1+and+ascii(lower(mid((select+pwd+from+users+limit+1,1),1,1)))=74
Signature Bypass	/?id=1+union+(select+'xz'from+xxx)

The reason why SQLIA are so popular and successful is because attackers have more opportunity to trick rule-based protection tools. Using a more sophisticated approach to attack detection like Machine Learning, can dramatically change the current situation.

3 Related Research

Machine learning techniques for SQL Injection Identification (SQLII) have become more popular in the last two decades. These tools use data generalization and allow for the discovery of new (unknown) threats by finding non-obvious dependencies in data sets.

Early works [3] purpose to analyze the query string and to find malicious patterns. This approach combines static analysis and runtime monitoring. Static analysis represents character-wise existing string analysis to extract from the web-application's code a model of all the query strings that could be generated by the application.

There is a group of tools that use Support Vector Machines [11][12]. These methods build decision boundaries in feature space which distinguish two (or more) classes of objects - legitime and crafted SQL query. The difference among many SQLII solutions is in different approaches to selecting characteristics for classification. For example, [13] where the SQL injection attacks (SQLIA) query is predicted with "injection points": SQL tokens, SQL symbols, disjoined text, and comments. These features allow creation of accurate machine learning classifiers based on Support Vector Machine.

Applying neural networks for classification of SQLIA type is implemented in [14]. The input for the neural network models are the SQLIA signatures. One-hot encoder is used for encoding signatures to the vector. The research results state that the proposed approach has high accuracy. However, using encoded signatures as an input feature vector decreases model generalization. Thus, this approach has the same limitations as rule-based WAF and IDS.

Another interesting approach is [15] and [16]. Authors perform known attack vectors as a template. Then similarity metrics between analyzing query and templates are estimated. Thus, the algorithm determines known SQLIA. The disadvantage of this method is that it is easy to bypass using various obfuscation schemes.

Deep learning models, like multilayer encoder-decoder are widely used for anomaly detection in text, and query analysis in particular. For instance, autoencoder is used for web application attack detection in paper [17]. Similar to previous methods, autoencoder requires exact matching of training data and testing data for successful attack detection.

Recurrent neural networks for SQLIA detection prospects are shown in [18]. The authors analyse the query as a sequence of tokens to predict the next element of a sequence. The conclusion is based on the difference between expected and actual tokens.

The group of unsupervised methods based on behaviour analysis are performed in [19]. It is presumed that a web application under attack acts different from regular operation. Thus, it is possible to detect leverage of SQL injection by anomaly detection tools.

Comparing these malicious pattern identification techniques in the query seems more effective as far as it allows prevention of the attack before the attacker starts to affect to the application.

The goal of this paper is to research advanced Machine Learning techniques for SQL injection identification and compare their results with IDS. We consider attack detection as a binary classification problem. We apply string analysis and natural language processing (NLP) to HTTP requests. Benign queries (including legal SQL requests) are labeled as a negative class. Malicious (positive) sets are represented by blind and error-based attack samples, UNION attacks, and obfuscation samples.

4 Experiments

The experiments include several steps described in General Scheme (see Fig. 2). A created data set from logged HTTP requests and open source attack samples is parsed and cleaned. The Feature Extraction allows us to convert the query value (text) to the vector using string analysis, bag-of-words, and word embedding techniques. This vector is an input to the classification model, that is capable of predicting if the query is malicious or benign.

4.1 General Scheme

The general scheme of our experiments is represented by graph in Fig. 2.

Data is extremely important for any Machine Learning task. Comprehensive data sets provide better problem generalization, which eventually provide consistency of classifiers with real world data. Features determine how raw data is represented to the model. As far as Machine Learning methods process numeric data, feature extraction means data vectorization. SQL Injection identification can be concerned with binary classification. We apply Machine Learning and Deep Learning in order to compare different model results. We use total accuracy, false positive rate, and prediction time as metrics to estimate model quality. Total accuracy is computed according to (1):

$$accuracy = \frac{TP + TN}{N} 100\%, \tag{1}$$

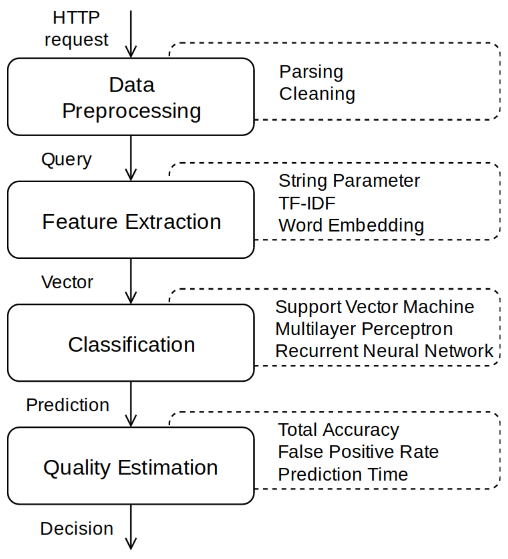


Figure 2: Experiment Scheme

where TP - true positive samples, TN - true negative samples, N - total number of samples.

False positive rate (FPR) is estimated as (2):

$$FPR = \frac{FP}{FP+TN}100\%, \qquad (2)$$

where FP - false positive samples.

The false positive rate is important in real security systems since false alerts reduce trust in security tools, which can cause problems when dealing with legitimate security alerts. We use prediction time as a parameter of model quality, as far as the proposed methods should be able to process requests in the order of millions per minute in real-time.

4.2 Data Acquisition and Preprocessing

The open access data of SQLIA [21] contains the templates of recognizable attack samples and does not include any samples of obfuscated requests. For that reason we created our own data set.

In order to get proper data covering different types of web applications, we analyzed data from various web applications like Wordpress web sites, JIRA, Confluence, e-shops, and education portals.

Benign data is taken from regular operation log files. We considered 276,000 unique negative samples in total. Alternatively to Uwagbole et.al at [13] and others, we use also valid SQL queries in the benign set. This reduces false positives alerts during model operation.

The positive (malicious) set was created using open data sets by recording attacks applying Open Source Penetration Testing Tools [20]. The open data set with SQL injection includes blind time-based, boolean-based, error-based attack samples, UNION attack samples for specific databases (MySQL, Oracle, PostgreSQL, SQLite, etc.), and active and passive attacks [22][23]. The HTTP generator [24] was used to get 312,000 unique samples for training and testing models. Input data was generated from web server logs using either the HTTP GET method (as in the browser address bar) or the POST method used in forms. Preprocessing steps include extracting URL, URL decode, parsing URL, query extracting, and clean-up data by deleting missing values.

Parsing the URL into components gives us the opportunity to analyze the most relevant information contained in the "parameter value" component. It is important to mention that some web applications use rewriting engines, e.g. Apache Module mod_rewrite, that perform URL as a parameter (path-123) instead of file system path as in Fig. 3.

4.3 Feature Extraction

We apply algorithmic feature extraction in string analysis, bag-of-words, and embedding as natural language processing vectorization.

4.3.1 String Analysis.

We considered the URL query as a simple string and represented it by the following feature sets:

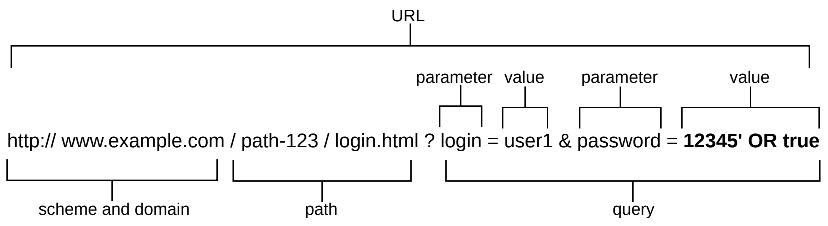

URL
parameter
value
parameter
value
http:// www.example.com / path-123 / login.html ? login = user1 & password = 12345' OR true
scheme and domain
path
query

Figure 3: URL Components

- countable parameters of query (total number of parameters and order of parameter in query);
- lexical characteristics of parameter values (string length, letters ratio, digits ratio, hexadecimal symbols ratio, and the white spaces ratio);
- keyword features based on frequency analysis;
- Punctuation symbol features based on frequency analysis.

First we considered the whole query and got total parameter count. Then we parsed the query into the parameter name - parameter value pairs (see Fig. 3) and got number (order) of the parameter in query.

We computed the lexical features for value: string length, letters ratio, digits ratio, hexadecimal symbols ratio, and the white spaces ratio. In addition to this list we applied frequency analysis for SQL command keywords and punctuation symbols. The idea is to evaluate the "maliciousness score" of the token (keyword or symbol), according to the formula:

$$score = \frac{f_{pos} - f_{neg}}{f_{pos} + f_{neg}}, \tag{3}$$

where f_{pos} - how often this token occurs in the positive set, f_{neg} - token frequency in the negative set.

The keywords set [25] were split into five groups based on maliciousness score: definitely malicious, likely malicious, unknown, popular, and benign. For example, the word "order" after normalization has frequency of 0.012 in positive set and 0.0014 in negative set. According to our formula (3) the maliciousness score is 0.8, meaning that this word belongs to malicious group. We applied the same approach to analyzing frequency of the punctuation symbols in the value string. This gave us three groups: malicious, unknown, and benign symbols. Then we counted the occurrence of the tokens from each group in value string.

Thus, we got a feature vector with 42 parameters in total. Selected features allow separate malicious and benign query pairing without overlapping. So we can use these vectors as an input for String Analysis models. Please find more details of experiment in section 4.4.

4.3.2 Natural Language Processing.

We considered two basic ways for query analysis that are applied in Natural Language Processing (NLP). These techniques consider sentences as a set of words or phrases. Words should be encoded to the vector form to be analyzed by a machine. There are several approaches to perform words as a vector. The first is known as bag-of-words (BoW). The idea of this algorithm is to represent sentences as a set of separate words and word occurrences in the data set. Each word from the whole data set (set of sentences) is written to the dictionary. The number of words in the dictionary determines length of feature vector. So, each word has its own position in this vector. Then we "replace" the word with its weight. The simplest way to compute this weight is to count word frequency. As result we have a sparse vector that contains word weights. It is important to note that this vector does not show the order of words in the sentence.

Applying BoW for SQL injection identification is based on special keywords and tokens [25]. We selected 52 of the most popular words that are used for SQLIA, for instance: SELECT, UNION, AND, OR, DROP, GROUP BY, and others. In addition, we split other words and symbols by character bigrams. For example, for a query like "?login=user1 OR a=a' " processed string looks as follow: ['us', 'se', 'er', 'r1', 'or', 'a=', =a', 'a"]. Then we vectorize the string with count frequency (CF) and token frequency - inverse document frequency technique (TF-IDF), and use these vectors as an input for the BoW classifier (see section 4.4).

The second vectorizing approach is called embedding and is used in sequential models like Recurrent Neural Networks. Unlike in bag-of-words, sequential algorithms allow analysis of words naturally, i.e. how they appear in the sentence. So information about word position and its context is preserved. We applied word embedding to create RNN input vector. This method allows presentation of words as a vector with a certain length in vector space so that the geometrical distance between words with similar meaning (synonyms) is minimal. The similarity of words is determined from their context (surrounding). During the string preprocessing step we created a dictionary consisting of 52 SQL keywords [25], 1513 character bigrams, and 12 one-grams with special characters used for comments and commands (–, =, ”). Then we split the initial string (parameter value) by tokens according to this dictionary. After vectorization, we got a sequence of vectors for the sequential models (see section 4.4).

4.4 Classification

Experiments are implemented using Scikit Learn Python [26] for data preprocessing and model prototyping, and Keras [27] for Deep Learning Models.

First, we considered a non-linear machine learning models based on string analysis data. We performed a feature vector as a model input. Output is a probability of class: if it is close to 0, the query is benign; if it close to 1, the query is malicious. We considered Support Vector Machine (SVM) and Multi-Layer Perceptron (MLP) in this task. According to Grid Search results the best parameters for SVM are: radial basis function with $C = 1000$ and $\gamma = 0.005$. According to our experiments, MLP has an architecture with 42 units (length of feature vector) in input layer, two hidden layers made of 100 and 50 units correspondingly, and two output units. We applied an ADAM optimizer for flexible adjustment of the learning rate.

Second, we created Neural Networks and used the sparse vector with bigram‘s weights as an inputs for this type of model. We used MLP with dropout layers for sparse vector analysis. It allows us to improve model performance by better generalizing and faster predicting. We applied Greed Search technique to find the best neural network architecture and hyperparameters. Architecture of MLP is as follows: 1521 units in the input layer (determined by bigram vocabulary size), 500 rectified linear units in the dense layer; 50% in the Dropout layer, 50 units in the dense layer, and two units it the output layer with logistic activation function.

Third, we researched two basic recurrent neurons that are used currently: Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). The core idea (main advantage) of these models is using previous state information to predict the current state. Concerning SQLII, we try to predict if the current token is a part of an attack based on its surroundings. The architecture of the sequentials model is: embedding layer (vocabulary size=1594, vector size=200), recurrent layer (LSTM or GRU) with 148 units corresponding to the maximum length of the input sequence; 50% dropout layer, two output units with sigmoid activation. Logistic function (sigmoid) has better performance for binary classification and allows output probability of classes.

5 Results

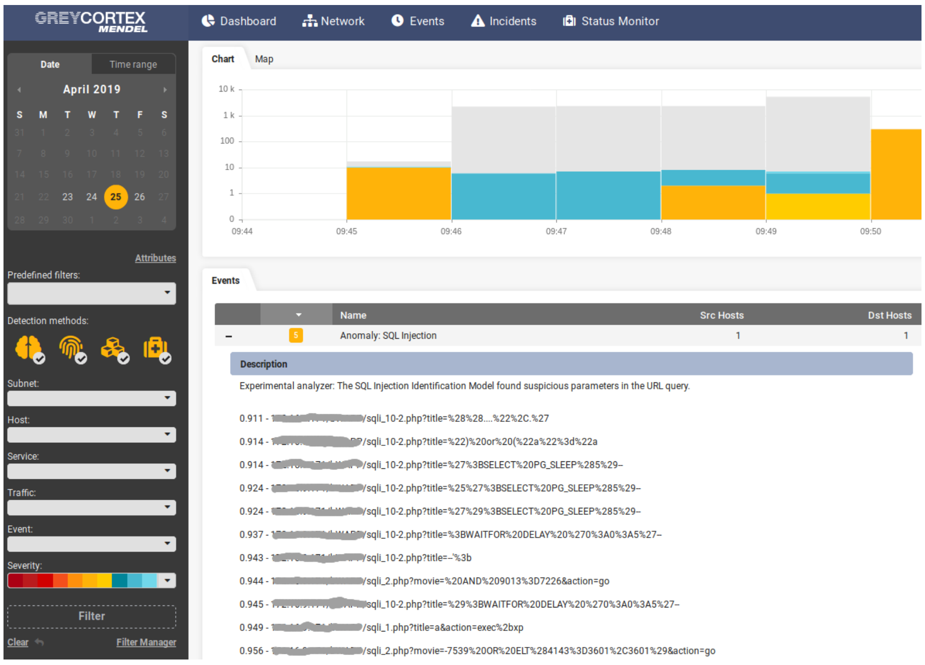
We used the validation data for model quality estimation. This data was not used during adjusting model training and parameter search. Applying a completely new data set allows us to predict how models work in real life. The validation data set contains 5260 positive and 5437 negative samples.

Table 2: Models and their quality estimations

Features	Model	Accuracy [%]	FP Rate [%]	Pred. Time [s]
Pattern Matching	IDS Rules	55.88	0.00	0.019
	L7 AD	73.52	0.00	0.212
String Analysis	SVM	99.59	0.43	0.035
	MLP	99.60	0.40	0.032
Bag of Words CF	MLP	99.70	0.27	0.259
Bag of Words TF-IDF	MLP	99.79	0.20	0.271
Embeddings	GRU	99.61	0.41	6.865
	LSTM	99.63	0.31	6.965

Table 2 contains results for classifiers tested with 10697 samples. As we can see, the reference Intrusion Detection System (IDS) has low accuracy and explains why we performed this research. The GREYCORTEX L7 Anomaly Detector (AD) has the feature that it learns the malicious L7 requests so it considers as anomalous just a limited number of incidents and it must be accomplished by a classifier. All of them show a high accuracy score in the validation set. However, using more sophisticated models increases the prediction time substantially.

Thus, the String Analysis with Multi-Layer Perceptron (MLP) has the best characteristics for practical deployment - the models from the first group are able to run as a part of Network Traffic Analysis for online traffic processing. We deployed the researched SQLII model within MENDEL, the network traffic analysis tool from GREYCORTEX, as a part of Application Layer (L7) security solution (see Fig. 4).



Events

		Name	Src Hosts	Dst Hosts
–	5	Anomaly: SQL Injection	1	1

Description

Experimental analyzer: The SQL Injection Identification Model found suspicious parameters in the URL query.

```
0.911 - [redacted]/sqli_10-2.php?title=%28%28....%22%2C.%27
0.914 - [redacted]/sqli_10-2.php?title=%22)%20or%20(%22a%22%3d%22a
0.914 - [redacted]/sqli_10-2.php?title=%27%3BSELECT%20PG_SLEEP%285%29–
0.924 - [redacted]/sqli_10-2.php?title=%25%27%3BSELECT%20PG_SLEEP%285%29–
0.924 - [redacted]/sqli_10-2.php?title=%27%29%3BSELECT%20PG_SLEEP%285%29–
0.937 - [redacted]/sqli_10-2.php?title=%3BWAITFOR%20DELAY%20%270%3A0%3A5%27–
0.943 - [redacted]/sqli_10-2.php?title=–'%3b
0.944 - [redacted]/sqli_2.php?movie=%20AND%209013%3D7226&action=go
0.945 - [redacted]sqli_10-2.php?title=%29%3BWAITFOR%20DELAY%20%270%3A0%3A5%27–
0.949 - [redacted]/sqli_1.php?title=a&action=exec%2bxp
0.956 - [redacted]/sqli_2.php?movie=-7539%20OR%20ELT%284143%3D3601%2C3601%29&action=go
```

Figure 4: MENDEL User Interface

6 Conclusion

We were focused on researching machine learning capabilities for the detection of malicious requests containing SQL Injection. SQLI identification was considered as a binary classification task with two classes: benign and malicious requests. String analysis, Bag of Words, and Embedding were performed as feature extracting to characterize parameters of query string. Support Vector Machine with Radial Basis Function, MultiLayer Perceptron, and Recurrent Neural Networks with Gated Recurrent Units (GRU) and Long Short-Term Memory (LSTM) were used as classification models.

We applied the same data sets consisting of approximately 600,000 samples for training and about 10,000 for testing different classifiers. We considered total accuracy, false positive rates, and prediction time as parameters of model quality. Then we compared model quality estimation on the validation sets. According to our results, all proposed models show high accuracy, but string analysis models work approximately 200 times faster than sequential models, mainly because of shorter feature vector.

In the future, we would like to develop models for detecting other important (L7) application attacks, e.g. NoSQL injections, command injections, Light-weight Directory Access Protocol (LDAP) injections, and cross-site scripting.

References

[1] OWASP Top 10 – 2017. 2017. The Ten Most Critical Web Application Security Risks. https://www.owasp.org/images/7/72/OWASP_Top_10-2017.pdf [Online; accessed 16-February-2019]

[2] Acunetix Wep Application Vulnerability Report. 2019. https://www.acunetix.com/blog/articles/acunetix-web-application-vulnerability-report-2019/ [Online; accessed 8-April-2019]

[3] Halfond, W. G. and Orso, A. 2005. AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks. In *20th IEEE/ACM International Conference on Automated Software Engineering*. ASE, pp. 174–183. DOI: 10.1145/1101908.1101935

[4] Kruegel, Ch. and Vigna, G. 2003. Anomaly Detection of Web-based Attacks. In *Proceedings of the ACM Conference on Computer and Communications Security*. ACM, Washington, DC, USA. DOI: 10.1145/948109.948144

[5] Justin, C. 2012. *SQL Injection Attacks and Defense*, second ed. Syngress Date, Elsevier.

[6] Dehariya, H., Shukla, P., and Ahirwar, M. 2016. A Survey on Detection and Prevention Techniques for SQL Injection Attacks. *International Journal of Wireless and Microwave Technologies* 6, pp. 72–79. DOI: 10.5815/ijwmt.2016.06.08

[7] Halfond, W. G., Viegas, J., and Orso, A. 2006. A Classification of SQL-Injection Attacks and Countermeasures. In *Proceedings of the IEEE International Symposium on Secure Software Engineering*. Vol 1, IEEE, pp. 13–15.

[8] Saidu Aliero, M., Aliyu Ardo, A., Ghani, I., and Atiku, M. 2016. Classification of Sql Injection Detection And Prevention Measure. *IOSR Journal of Engineering* 6, pp. 06–17.

[9] Alnabulsi, H., Islam, Md R., Mamun, Q. 2014. Detecting SQL injection attacks using SNORT IDS. In *Asia-Pacific World Congress on Computer Science and Engineering.* No. 14968012, IEEE. DOI: 10.1109/APWCCSE.2014.7053873

[10] SQL Injection Bypassing WAF. 2017. https://www.owasp.org/index.php/SQL_Injection_Bypassingh_WAF. [Online; accessed 16-February-2019]

[11] Ladole, A. and Phalke, M.D. 2016. SQL Injection Attack and User Behavior Detection by Using Query Tree, Fisher Score and SVM Classification. *International Research Journal of Engineering and Technology* 3, 6, pp. 1505–1509.

[12] Kar, D., Panigrahi, S., and Sundararajan, S. 2016. SQLiGoT: Detecting SQL injection attacks using graph of tokens and SVM. *Computers & Security* 60, pp. 206–225. DOI: 10.1016/j.cose.2016.04.005

[13] Uwagbole, S., Buchanan, W., and Fan, L. 2017. Applied Machine Learning Predictive Analytics to SQL Injection Attack Detection and Prevention. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM).* No. 17058611, IEEE. DOI: 0.23919/INM.2017.7987433

[14] Moradpoor, N. 2015. SQL-IDS: Evaluation of SQLi Attack Detection and Classification Based on Machine Learning Techniques. In *SIN '15 Proceedings of the 8th International Conference on Security of Information and Networks.* ACM, Sochi, Russia, pp. 258–266.

[15] Kar, D., Panigrahi, S., and Sundararajan, S. 2015. SQLiDDS: SQL Injection Detection using Query Transformation and Document Similarity. In *International Conference on Distributed Computing and Internet Technology.* Lecture Notes in Computer Science, vol 8956. Springer, pp. 377–390. DOI: 10.1007/978-3-319-14977-641

[16] Kar, D., Panigrahi, S., and Sundararajan, S. 2016. SQLiDDS: SQL injection detection using document similarity measure. *Journal of Computer Security* 24, pp. 507–539. DOI: 10.3233/JCS-160554

[17] Murzina A. and Stepanyuk I. 2019. Detecting Web Attacks with a Seq2Seq Autoencoder https://blog.ptsecurity.com/2019/02/detecting-web-attacks-with-seq2seq.html [Online; accessed 21-January-2019].

[18] Skaruz, J. and Seredyński, F. 2007. Recurrent neural networks towards detection of SQL attacks. In *2007 IEEE International Parallel and Distributed Processing Symposium.* No. 9516781, IEEE, pp. 1–8. DOI: 10.1109/IPDPS.2007.370428

[19] Mirsky, Y., Doitshman, T., Elovici, Y., Shabtai, and A. Kitsune. 2018. An Ensemble of Autoencoders for Online Network Intrusion Detection. In *Network and Distributed System Security Symposium.* DOI: 10.14722/ndss.2018.23211

[20] List of Best Open Source SQL Injection Tools. 2018. https://kalilinuxtutorials.com/sql-injection/. [Online; accessed 21-January-2019]]

[21] Park, S. 2017. Machine Learning. GitHub Repository https://github.com/Scott-Park/MachineLearning/tree/master/Sql-Injection/source/trainingdata [Online; accessed 15-January-2019].

[22] Wylie, B. 2014. SQL Injection. GitHub Repository https://github.com/SuperCowPowers/data_hacking/tree/master/sql_injection/data [Online; accessed 15-January-2019].

[23] FuzzDB Project. 2016. SQL Injection. GitHub Repository https://github.com/fuzzdb-project/fuzzdb/tree/master/attack/sql-injection [Online; accessed 16-January-2019].

[24] Fujdiak, R., Uher, V., Mlynek, P., et al. 2018. IP Traffic Generator Using Container Virtualization Technology. In *2018 10th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT).* Moscow, Russia. DOI: 10.1109/ICUMT.2018.8631248

[25] Basta, C., Elfatatry, A., and Darwish, S. 2016. Detection of SQL Injection Using a Genetic Fuzzy Classifier System. *International Journal of Advanced Computer Science and Applications* 7, 6, pp. 129–137. DOI: 10.14569/IJACSA.2016.070616

[26] Pedregosa et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12, pp. 2825–2830.

[27] Chollet, F. et al. 2015. Keras. https://keras.io.