Annales UMCS Informatica AI 6 (2007) 37-48

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

Recurrent neural networks in the context of SQL attacks

Jarosław Skaruz[1*], Franciszek Seredyński[2]

[1]*Institute of Computer Science, University of Podlasie, Sienkiewicza 51, 08-110 Siedlce, Poland*
[2]*Polish-Japanese Institute of Information Technology, Koszykowa 86, 02-008 Warszawa,*
Institute of Computer Science, Polish Academy of Sciences, Ordona 21, 01-237 Warszawa, Poland

Abstract

In the paper we present a new approach based on application of neural networks to detect SQL attacks. SQL attacks are those attacks that take advantage of using SQL statements to be performed. The problem of detection of this class of attacks is transformed to time series prediction problem. SQL queries are used as a source of events in a protected environment. To differentiate between normal SQL queries and those sent by an attacker, we divide SQL statements into tokens and pass them to our detection system, which predicts the next token, taking into account previously seen tokens. In the learning phase tokens are passed to recurrent neural network (RNN) trained by backpropagation through time (BPTT) algorithm. Teaching data are shifted by one token forward in time with relation to input. The purpose of the testing phase is to predict the next token in the sequence. All experiments were conducted on Jordan and Elman networks using data gathered from PHP Nuke portal. The experimental results show that the Jordan network outperforms the Elman network predicting correctly queries of the length up to ten.

1. Introduction

Integrity, confidentiality and availability are the main features of computer security. A large number of Web applications, especially those deployed for companies to e-business purpose must meet these requirements. Such applications are written in script languages like PHP embedded in HTML allowing to establish connection to databases, retrieving data and putting them in WWW site. Besides that all Web contents is often based on the retrieved data, a database also stores sensitive user typed data like credit card numbers and personal information. Security violations consist in not authorized access and modification of data in the database. SQL is one of the languages used to manage data in databases. Its statements can be one of sources of events for potential attacks. One of the ideas to detect an intruder using SQL statements is

[*]Corresponding author: *e-mail address*: Jaroslaw.Skaruz@ap.siedlce.pl

to build a profile of normal behaviour and in detection stage compare it with the observed events.

In literature there are some approaches to intrusion detection in Web applications. In [1] the authors developed an anomaly-based system that learns the profiles of the normal database access performed by web-based applications using a number of different models. A profile is a set of models, to which parts of SQL statement are fed to in order to train the set of models or to generate an anomaly score. During training phase models are built based on training data and anomaly score is calculated. For each model, the maximum of anomaly score is stored and used to set an anomaly threshold. During the detection phase, for each SQL query anomaly score is calculated. If it exceeds the maximum of anomaly score evaluated during the training phase, the query is considered to be anomalous. Decreasing false positive alerts involves creating models for custom data types for each application to which this system is applied.

Besides that work, there are some other works on detecting attacks on a Web server which constitutes a part of infrastructure for Web applications. In [2] detection system correlates the server-side programs referenced by clients queries with the parameters contained in these queries. It is a similar approach to detection to the previous work. The system analyzes HTTP requests and builds a data model based on attribute length of requests, attribute character distribution, structural inference and attribute order. In a detection phase the built model is used for comparing requests of clients.

In [3] logs of Web server are analyzed to look for security violations. However, the proposed system is prone to high rates of false alarm. To decrease it, some site-specific available information should be taken into account which is not portable.

In this work we present a new approach to intrusion detection in Web application. Rather than building profiles of normal behaviour we focus on a sequence of tokens within SQL statements observed during normal use of application. Two architectures of RNN are used to encode stream of such SQL statements.

The paper is organized as follows. The next section discusses SQL attacks. In section 3 we present two architectures of RNN. Section 4 shows training and testing data used for experiments. Next, section 5 contains experimental results. Last section summarizes results and shows possible future work.

2. SQL Attacks

2.1. SQL injection

SQL injection attack consists in such a manipulation of an application communicating with a database, that it allows a user to gain access or to allow it

to modify data for which it has not privileges. To perform an attack in most cases the Web forms are used to inject part of SQL query. Typing SQL keywords and control signs an intruder is able to change the structure of SQL query developed by a Web designer. It is possible because parts of SQL statements depend on the data typed by a user. If variables used in the SQL query are under control of a user, he can modify the SQL query which will cause change of its meaning. Consider an example of a poor quality code written in PHP presented below.

```
$connection=mysql_connect();
mysql_select_db("test");
$user=$HTTP_GET_VARS['username'];
$pass=$HTTP_GET_VARS['password'];
$query="select * from users where login='$user' and password='$pass'";
$result=mysql_query($query);
if(mysql_num_rows($result)==1)  echo "authorization successful"
else   echo "authorization failed";
```

The code is responsible for authorizing users. User data typed in a Web form are assigned to variables *user* and *pass* and then passed to the SQL statement. If retrieved data include one row it means that a user filled in the form login and password the same as stored in the database. Because data sent by a Web form are not analyzed, a user is free to inject any strings. For example, an intruder can type: "' *or 1=1* --" in the login field leaving the password field empty. The structure of SQL query will be changed as presented below.

```
$query="select * from users where login='' or 1=1 --' and password='';
```

Two dashes comment the following text. The boolean expression *1=1* is always true and as a result the user will be logged with privileges of the first user stored in the table *users*.

2.2. Proposed approach

The way we detect intruders can be easily transformed to a time series prediction problem. According to [4] a time series is a sequence of data collected from some system by sampling a system property, usually at regular time intervals. One of the goals of the analysis of time series is to forecast the next value in the sequence based on the values occurred in the past. The problem can be more precisely formulated as follows:

$$S_{t-2}, S_{t-1}, S_t \rightarrow S_{t+1}, \tag{1}$$

where S is any signal, which is dependent on a solving problem and t is a current moment in time. Given S_{t-2}, S_{t-1}, S_t, we want to predict S_{t+1}. In the problem of

detection of SQL attacks, each SQL statement is divided into some signals, which we further call tokens. The idea of detecting SQL attacks is based on their key feature. The SQL injection attacks involve modification of SQL statement, which lead to the fact that the sequence of tokens extracted from a modified SQL statement is different from that derived from a legal SQL statement. For example, let *S* mean the recorded SQL statement and T_1, T_2, T_3, T_4, T_5, tokens of this SQL statement. The original sequence of tokens is as follows:

$$T_1, T_2, T_3, T_4, T_5 . \tag{2}$$

If an intruder performs an attack, the form of SQL statement changes. Transformation of the modified statement to tokens results in different tokens from those shown in eq. (2). The example of a sequence of tokens related to the modified SQL query is as follows:

$$T_1, T_2, T_{mod3}, T_{mod4}, T_{mod5} . \tag{3}$$

Tokens number 3, 4, 5 are modified due to an intruder activity. We assume that the intrusion detection system trained on the original SQL statements is able to predict the next token based on those from the past. If the token T_1 occurs, the system should predict token T_2, next token T_3 is expected. In the case of attacks token T_{mod3} occurs which is different from T_3, which means that an attack is performed.

Various techniques have been used to analyze time series [5,6]. Besides statistical methods, RNNs have been widely used for that problem. In our study presented in this paper we selected two RNNs, the Elman and the Jordan networks.

3. Recurrent Neural Networks

3.1. General issues

In comparison to feedforward neural networks RNN have feedback connections which provide dynamics. When they process information, output neurons signal depends on input and activation of neurons in the previous steps of teaching RNN. However, RNNs suffer from *vanishing gradient* [7]. This is the main reason why gradient-descent algorithms are not sufficiently powerful to map the relationship between the output of RNN and the input that occur much earlier in time. In [7] the authors compared the Elman network with the neural one based on the NARX model. The model assumes that the output neuron signals from *n* times in back are passed to the hidden layer neurons. This partially overcomes *the vanishing gradient* effect. Some researchers introduce modifications to known architectures of RNN to improve the teaching process. In [8] the additional self-feedback connection to the context layer neurons of the Elman network was added. The experimental results show that the error of

network when weight of additional connection is fixed is smaller than that of the Elman network.

3.2. RNN architectures

There are some differences between the Elman and the Jordan networks. The first is that the input signal for the context layer neurons comes from different layers and the second is that the Jordan network has additional feedback connection in the context layer. While in the Elman network the size of the context layer is the same as the size of the hidden layer, in the Jordan network the size of output layer and context layer is the same. In both networks recurrent connections have fixed weight equal to 1.0. The networks were trained by BPTT and the following equations are applied for the Elman network:

$$\mathrm{x}(\mathrm{k}) = \left[\mathrm{x}_1(\mathrm{k}),...,\mathrm{x}_N(\mathrm{k}),\mathrm{v}_1(\mathrm{k\text{-}1})\right], \tag{4}$$

$$u_j(k) = \sum_{i=1}^{N+K} \mathrm{w}_{\mathrm{ij}}^{(1)} \mathrm{x}_\mathrm{i}(\mathrm{k}), \qquad \mathrm{v}_\mathrm{j}(\mathrm{k}) = \mathrm{f}\left(\mathrm{u}_\mathrm{j}(\mathrm{k})\right), \tag{5}$$

$$g_j(k) = \sum_{i=1}^{K} \mathrm{w}_{\mathrm{ij}}^{(2)} \mathrm{v}_\mathrm{i}(\mathrm{k}), \qquad \mathrm{y}_\mathrm{j}(\mathrm{k}) = \mathrm{f}\left(\mathrm{g}_\mathrm{j}(\mathrm{k})\right), \tag{6}$$

$$E(k) = 0.5\sum_{i=1}^{M}\left[y_i(k) - d_i(k)\right]^2, \tag{7}$$

$$\delta_i^{(o)}(k) = [\left[y_i(k) - d_i(k)\right] f'\left(g_i(k)\right), \tag{8}$$

$$\delta_i^{(h)}(k) = f'\left(u_i(k)\right)\sum_{j=1}^{M}\delta_j^{(o)}(k) w_{ij}^{(2)}, \tag{9}$$

$$w_{ij}(k+1)^{(2)} = w_{ij}(k)^{(2)} + \sum_{k=1}^{sql\_length}\left[v_i(k)\delta_j^{(o)}(k)\right], \tag{10}$$

$$w_{ij}(k+1)^{(1)} = w_{ij}(k)^{(1)} + \sum_{k=1}^{sql\_length}\left[x_i(k)\delta_j^{(h)}(k)\right]. \tag{11}$$

In equations (4)-(11), N, K, M stand for the size of the input, hidden and output layers, respectively. x(k) is an input vector, $u_j(k)$ and $g_j(k)$ are input signals provided to the hidden and output layer neurons. Next, $v_j(k)$ and $y_j(k)$ stand for the activations of the neurons in the hidden and output layers at time *k*, respectively. Equation (7) shows how RNN error is computed, while neuron errors in the output and hidden layers are evaluated according to (8) and (9), respectively. Finally, in the last step values of weights are changed using formulas (10) for the output layer and (11) for the hidden layer.

3.3. Training

The training process of RNN is performed as follows. The tokens of the SQL statement become input of a network. Activations of all neurons are computed. Next, an error of each neuron is calculated. These steps are repeated until the last token has been presented to the network. Next, all weights are evaluated and activation of the context layer neurons is set to 0. For each input data, teaching data are shifted by one token forward in time with relation to the input. Training data consist of 276 SQL queries without repetition. The following tokens are considered: keywords of SQL language, numbers, strings and combinations of these elements. We used the collection of SQL statements to define 54 distinct tokens. Each token has a unique index. Table 1 shows the selected tokens and their indices.

Table 1. A part of a list of tokens and their indexes

Token	Index
…	…
WHERE	7
FROM string	28
SELECT string	36
String=number	47
INSERT INTO	54
…	…

The indices are used for preparation of input data for neural networks. The index e.g. of a keyword *WHERE* is 7. The index 28 points to a combination of keyword *FROM* and any string. The token with index 36 relates to a grammatical link between *SELECT* and any string. Finally, when any string is compared to any number within a SQL query, the index of a token equals to 47. Figure 1 presents an example of SQL statement, its representation in the form of tokens and related binary four inputs of a network.

SQL statement is encoded as k vectors, where k is the number of tokens constituting the statement (see figure 1). The number of neurons on the input layer is the same as the number of defined tokens. Networks have 55 neurons in the output layer. 54 neurons correspond to each token similarly to the input layer but the neuron 55 is included to indicate that just processing input data vector is the last within a SQL query. Training data, which are compared to the output of the network have the value either equal to 0.1 or 0.9. If a neuron number n in the output layer has small value then it means that the next processing token can not have index n. On the other hand, if output neuron number n has the value of 0.9, then the next token in a sequence should have index n.

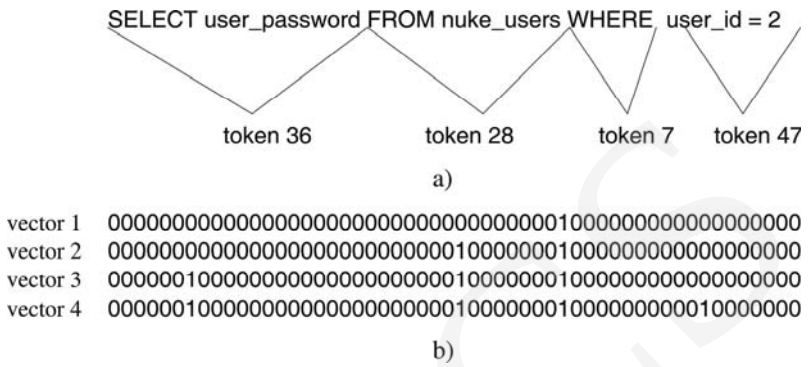

Fig. 1. Preparation of input data for a neural network: analysis of a statement in terms of tokens (a), input neural network data corresponding to the statement (b)

At the beginning, the SQL statement is divided into tokens. The indices of tokens are: 36, 28, 7 and 47. Each row is an input vector for RNN. In Figure 1 the first token that has appeared is 36. As a consequence, in the first step of training the output signal of all neurons in the input layer is 0 except neuron number 36, which has the value of 1. Next input vectors indicate current indices of tokens and the index of a token that has been processed by RNN. The next token in a sequence has the index equal to 28. It follows that only neurons 36 and 28 have the output signal equal to 1. The next index of a token is 7, which means that neurons: 36, 28 and 7 send 1 and all remaining neurons send 0. Finally, neurons 36, 28, 7, 47 have the activation signal equal to 1. At that moment weights of RNN are updated and the next SQL statement is considered.

4. Training and testing data

We evaluated our system using the data collected from PHP Nuke portal [9]. It is a well known application with many holes in older versions. Similarly to [1] we installed this portal in version 7.5, which is susceptible to some SQL injection attacks. A function of the portal related to executing the SQL statements was modified. Besides its original purpose, each executed SQL query is written to a log file. Data without attacks were gathered by visiting the Web sites using a browser. Each time a Website is downloaded by a browser either a link is clicked or filled forms are executed, SQL queries are sent to a database and logged to a file simultaneously. The set of all SQL queries was divided into 12 subsets, each containing SQL statements of different lengths. 80% of each data set were used for training and the remaining data were used for examining generalization. Teaching data are shifted one time forward in time. The data with attacks are the same as reported in [1].

5. Experimental results

Experimental study was divided into four stages. In the first one, we wanted to evaluate the best parameters of both RNNs and learning algorithm. These features are: a number of neurons in the hidden layer, α used in momentum, activation function of neurons in the hidden and output layers, η that determines the extent of weights update.

For the Elman network all neurons in the hidden layer have the sigmoidal activation function while all neurons in the output layer have the *tanh* function. For the Jordan network the *tanh* function was chosen for the hidden layer and the sigmoidal function for the output layer. For each data subset we run RNNs 10 times for various initial values of weights, specifying a different value of η and a constant value of α. Next, having the best value of η we modified α parameter. These tests were repeated for all combinations of settings of function activation for the hidden and output layer neurons. The number of neurons in the hidden layer was also evaluated during experiments – for 58 neurons the RMS error was minimal. In 75% cases η does not exceed 0.2 and α value in 87.5% of experiments is less than 0.2. In the second phase of the experimental study we trained 12 RNNs, one for each training data subset, using the values from the first stage. Figure 2 shows how error of both networks changes through epochs.

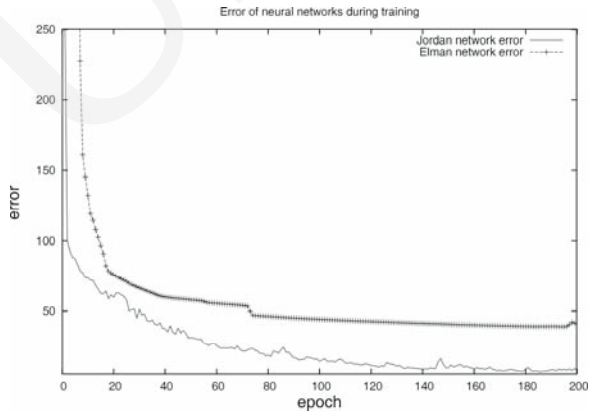

Fig. 2. Neural networks training process for the data subset with 10-length SQL queries

The input for both networks were the subset containing sequences of the length equal to 10. From the beginning of the training, the error of the Jordan network was much smaller than that of the Elman network. In the next few epochs the error of both networks decreased quickly but the Jordan network error remained much smaller than the Elman network one.

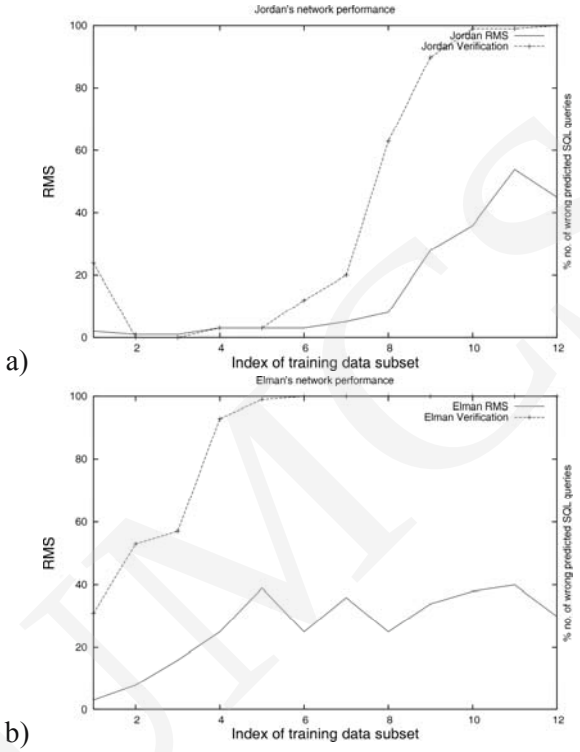

Fig. 3. Error and number of wrongly predicted SQL queries for each subset of data: Jordan network (a), Elman network (b)

Figure 3 (a) and (b) shows how error of networks changes for all subsets of SQL queries. The figure also depicts how well the networks are verified. Here, a statement is considered and predicted if for all input vectors, all neurons in the output layer have the values according to the training data. All values presented in the figure are averaged on 10 runs of RNNs. One can see that nearly for all data subsets the Jordan network outperforms the Elman one. Only for data subsets 11 and 12 (see Table 2) the error of the Jordan network is greater than that of the Elman network. Despite this fact for all data subsets the percentage number of wrongly predicted SQL queries for the Jordan network is less than that of wrongly predicted SQL statements for the Elman network. The number of input vectors depends on the number of SQL queries and the length of SQL query. The Jordan network is able to predict all tokens of 10 length statements (20.6% false alarms).

In the third part of experiments we checked if RNNs correctly detect attacks. Each experiment was conducted using trained RNNs from the second stage. Figure 4a) presents the typical RNN output if an attack is performed. The left

column depicts the number of input vector for RNN, while the right column shows the number of cases in which the index of the token indicated by the network output is different from that of the next processed by RNN token. It is common for each network, that nearly each output vector of a network has a few errors. This phenomenon is present for all attacks used in this work. Based on that observation, a decision about good or bad verification and generalization of a network can be taken in the correlation with a form of network output against attacks. Figure 4b) shows RNN output for SQL statements derived from the training set and that, which was not present in the training set. The description of figure 4b) is similar to that of figure 4a). The second column relates to the case if the SQL query was in the training set and the third column concerns the SQL query, which was not in the training set.

# SQL statement	
# index of input vector	number of errors
1	7
2	2
3	1
4	2
5	2
6	1
7	2
8	0

a)

# SQL statement		
# index of input vector	num. of errors- ver	num. of errors- gen
1	0	0
2	1	1
3	1	2
4	0	1
5	0	1
6	1	1
7	1	1
8	1	0

b)

Fig. 4. RNN output for an attack (a), RNN output for known and unknown SQL statement

The number of errors during verification and generalization is much smaller than that when an attack is processed by RNN. Moreover, there are also more output vectors free of errors. An easily noticeable difference between an attack and a normal activity allows us to re-evaluate the obtained results presented in Figure 4. To distinguish between an attack and a legitimate SQL statement we define the following rule for the Jordan network: an attack occurred if the average number of errors for each output vector is not less than 2.0 and 80% of output vectors include any error. When the Elman network is used, the threshold equals to 1.6 and the percentage of output vectors possessing errors equals to 90%. Applying these rules ensures that all attacks are detected by both RNN. Table 2 presents the percentage number of SQL statements wrongly predicted during verification and generalization if the results were processed by the rules. The ranges 2-4, 13-14, 15-16 and 17-20 of the data subset (see Table 2) mean that these subsets include SQL queries of the lengths between 2-4, 13-14, 15-16, 17-20. All remaining subsets contain fixed length statements.

Table 2. Results of verification and generalization of Elman and Jordan networks

Data subset	Elman ver	Elman gen	Jordan ver	Jordan gen
2-4	0	0	0	0
5	0	1.4	0	0
6	0	24.2	0	12.8
7	0	15.7	0	1.4
8	0	5	0	1.6
9	0	3.33	0	0
10	0	2.5	0	0
11	0	10	0	13.33
12	0	0	0	6.66
13-14	0	0	0	0
15-16	0	3.33	0	3.33
17-20	0	40	0	13.33

For most cases the Jordan network outperforms the Elman network. Only for the data subsets containing statements made from 11 and 12 tokens, the Elman network is a little better than the Jordan network. The important outcome of defined rules is that both RNNs thought all statements and only a few legitimate statements, which were not in the training set were detected as attacks.

Conclusions

In the paper we have presented a new approach to detecting SQL-based attacks. The problem of detection was transformed to the time series prediction problem and two RNNs were examined to show their potential use for such a class of attacks. Profound analysis of the experimental results leads to the definition of rules used for distinguishing between an attack and a legitimate statement. When these rules are applied, both networks are completely trained for all SQL queries included in all the training subsets. Accuracy of the results very strongly depends on the rules. The advisable part of experimental study is to apply the defined rules to the other data set, which can confirm efficiency of the proposed approach to detecting SQL attacks.

References

[1] Valeur F., Mutz D., Vigna G., A *Learning-Based Approach to the Detection of SQL Attacks*, Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment, Austria, (2005).

[2] Kruegel C., Vigna G., *Anomaly Detection of Web-based Attacks*. Proceedings of the 10th ACM Conference on Computer and Communication Security (CCS '03), (2003) 251.

[3] Almgren M., Debar H., Dacier M., *A lightweight tool for detecting web server attacks*. In Proceedings of the ISOC Symposium on Network and Distributed Systems Security, (2000).

[4] Nunn I., *The Application of Antigenic Search Techniques to Time Series Forecasting*. In Proceedings of the Genetic and Evolutionary Computation Conference, USA, (2005).
[5] Kendall M., Ord J., *Time Series*, third edition, (1999).
[6] Pollock D., *A Handbook of Time-Series Analysis*. Signal Processing and Dynamics, Academic Press, London, (1999)
[7] Lin T., Horne B.G., Tino P., Giles C.L., *Learning long-term dependencies in NARX recurrent neural networks*. IEEE Transactions on Neural Networks, (1996) 1329.
[8] Drake P.R., Miller K.A., *Improved Self-Feedback Gain in the Context Layer of a Modified Elman Neural Network*. Mathematical and Computer Modelling of Dynamical Systems, (2002) 307.
[9] http://phpnuke.org/