Recurrent neural networks towards detection of SQL attacks

Jaroslaw Skaruz[1], Franciszek Seredynski[1,2,3]

[1]Institute of Computer Science
University of Podlasie
Sienkiewicza 51, 08-110 Siedlce Poland
jaroslaw.skaruz@ap.siedlce.pl

[2]Institute of Computer Science
Polish Academy of Sciences
Ordona 21, 01-237 Warsaw Poland
sered@ipipan.waw.pl

[3]Polish-Japanese Institute of Information Technology
Koszykowa 86, 02-008 Warsaw 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. 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. 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 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 to build a profile of normal behavior and in detection stage compare it with observed events.

In the literature there are some approaches to intrusion detection in Web applications. In [1] the authors developed 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 detection phase, for each SQL query anomaly score is calculated. If it exceeds the maximum of anomaly score evaluated during 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 similar approach to detection to the previous work. The sys-

1-4244-0910-1/07/$20.00

tem analyzes HTTP requests and builds data model based on attribute length of requests, attribute character distribution, structural inference and attribute order. In a detection phase 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. Tan et al. [4] discuss some hints for intruders to be not detected by intrusion detection systems.The study shows weakness of intrusion detectors and shows how an attacker can effectively modify common exploits to take advantage of those weakness in order to craft an offensive mechanism that renders an anomaly-based intrusion detector blind to the on-going presence of those attacks.

In this work we present a new approach to intrusion detection in Web application. Rather than building profiles of normal behavior 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 the most cases 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 SQL query are under control of a user, he can modify 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 comments the following text. Boolean expression *1=1* is always true and as a result user will be logged with privileges of the first user stored in the table *users*.

2.2 Cross Site Scripting

Cross Site Scripting known as XSS is another class of attacks on Web applications, which through inserting malicious data into a database can cause hijacking network connection of a privileged user. Often Web sites includes scripts written in JavaScript or in VBScript embedded in HTML, which executes on a client side making application more functional providing mechanisms for checking correctness of data types in forms or executing any function. Attackers exploit trust relationship between a Web server and a browser. This class of attack can occur when data sent to the server are not checked and then they are downloaded by other users and put into web site. If the data typed in a form is a malicious script, it will be executed by a victim's browser. In the simplest case, a user will be shown pop-up window with his *session id*, which completely identifies a user. The code below is responsible for such an action.

```
<script>alert(document.cookie);</script>
```

When the script is retrieved from the server it is parsed and executed by a browser. A window with contents of a cookie is displayed to a user. Because string identificating a user is stored in a cookie, stealing cookie leads to session hijacking allowing an intruder to masquerade as a victim.

2.3 Other Attacks

Common architecture of a software needs application communicating with a database, which purpose is to deliver data to a user software. The software can be written in any programming languages like C or Java with graphical user

interface different than WWW sites. Because once installed software uses constant set of SQL statements, each occurrence of different than previously recorded SQL statements can be treated as threatening. One of the causes of their presence can be an intruder activity. On the other hand if a new software is deployed, some new SQL statements will be sent to a database. This time these statements are legal. The presence of threatening SQL statements should be detected. Our approach to detection SQL attacks described in the next section can be applied to all discussed classes of attacks.

2.4 Proposed approach

The way we detect intruders can be easily transformed to time series prediction problem. According to [5] 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 goal of the analysis of time series is to forecast the next value in the sequence based on values occurred in the past. The problem can be more precisely formulated as follows:

$$s_{t-2}, s_{t-1}, s_t \longrightarrow s_{t+1}, \tag{1}$$

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 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. SQL injection and XSS attacks involve modification of SQL statement, which lead to the fact, that the sequence of tokens extracted from a modified SQL statement is different than the sequence of tokens derived from a legal SQL statement. For example, let *S* means 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 than these shown in eq.(2). The example of a sequence of tokens related to 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 intrusion detection system trained on original SQL statements is able to predict the next token based on the tokens from the past. If the token T_1 occurs, the system should predict token T_2, next token T_3 is expected. In case of attacks token T_{mod3} occurs which is different than T_3, which means that an attack is performed.

Various techniques have been used to analyze time series [6, 7]. 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

Application of neural networks to solving any problem involves three steps. The first is training, during which weights of network connections are changed. Network output is compared to training data and the network error is evaluated. In the second step the network is verified. Values of connections weights are constant and the network is checked if its output is the same as in the training phase. The last step is generalization. The network output is evaluated for such data, which were not used for training the network. Good generalization is a desirable feature of all networks because it means that the network is prepared for processing data, which may occur in the future.

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 *vanishing gradient* [8]. This is the main reason why gradient-descent algorithms are not sufficiently powerful to map relationship between output of RNN and input that occur much earlier in time. In [8] the authors compared the Elman network with neural network based on NARX model. The model assumes that output neuron signals from *n* times in back are passed to the hidden layer neurons. This partially overcome *vanishing gradient* effect. Some researchers introduce modifications to known architectures of RNN to improve teaching process. In [9] additional self-feedback connection to context layer neurons of the Elman network was added. Experimental results show that error of network when weight of additional connection is fixed is smaller than error of the Elman network. In the next section we show two architectures of RNNs presented in figures 1 and 2.

3.2 RNN architectures

There are some differences between the Elman and the Jordan networks. The first is that input signal for context layer neurons comes from different layers and the second is that 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. Networks were trained by

BPTT and the following equations are applied for the Elman network:

$$x(k) = [x_1(k), ..., x_N(k), v_1(k-1), ..., v_K(k-1)], \quad (4)$$

$$u_j(k) = \sum_{i=1}^{N+K} w_{ij}^{(1)} x_i(k), v_j(k) = f(u_j(k)), \quad (5)$$

$$g_j(k) = \sum_{i=1}^{K} w_{ij}^{(2)} v_i(k), y_j(k) = f(g_j(k)), \quad (6)$$

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

$$\delta_i^{(o)}(k) = [y_i(k) - d_i(k)] f^{`}(g_i(k)), \quad (8)$$

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

$$w_{ij}(k+1)^{(2)} = w_{ij}(k)^{(2)} + \sum_{k=1}^{sql-length} [v_i(k) \delta_j^{(o)}(k)], \quad (10)$$

$$w_{ij}(k+1)^{(1)} = w_{ij}(k)^{(1)} + \sum_{k=1}^{sql-length} [x_i(k) \delta_j^{(h)}(k)]. \quad (11)$$

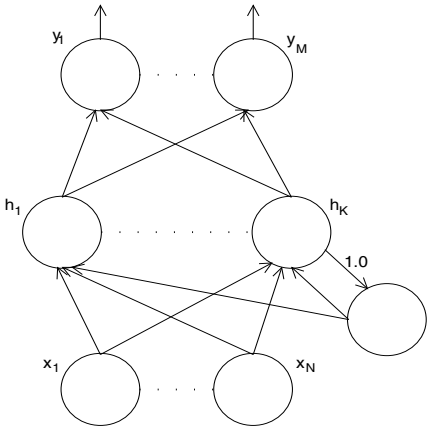

Figure 1. Elman network

In the 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 layer at time *k*, respectively. The equation (7) shows how RNN error is computed, while neurons error 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.

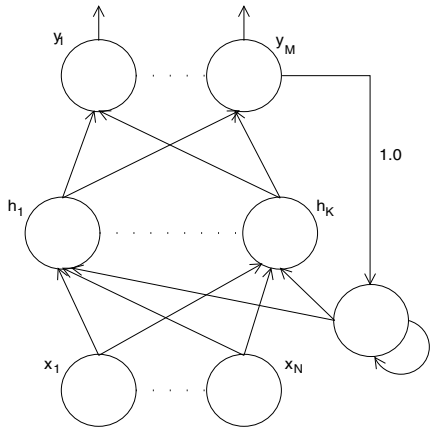

Figure 2. Jordan network

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

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 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 input. Training a network in such a way ensures that it will posses prediction capability.

Training data consists 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. The table 1 shows selected tokens and their indexes. The indexes 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 3 presents an example of SQL statement, its representation in the form of tokens and related binary four inputs of a network. SQL

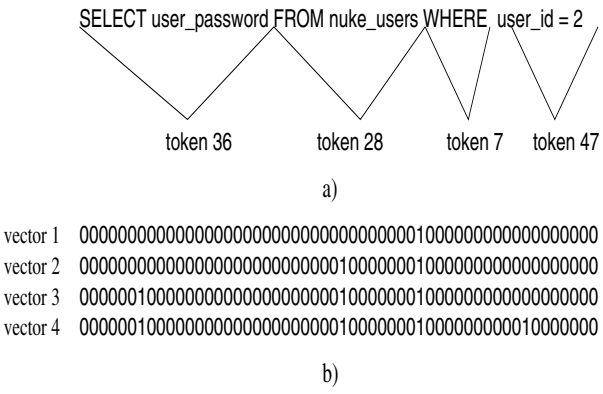

Figure 3. Preparation of input data for a neural network: analysis of a statement in terms of tokens (a), input neural network data corresponding to the statement (b)

statement is encoded as k vectors, where k is the number of tokens constituting the statement (see figure 3). 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 value either equals 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 value of 0.9, then the next token in a sequence should have index n.

At the beginning, SQL statement is divided into tokens. The indexes of tokens are: 36, 28, 7 and 47. Each row is an input vector for RNN (see figure 3). In the figure 3 the first token that has appeared is 36. As a consequence, in the first step of training output signal of all neurons in the input layer is 0 except neuron number 36, which has value of 1. Next input vectors indicate current indexes of tokens and the index of a token that has been processed by RNN. The next token in a sequence has index equals to 28. It follows that only neurons 36 and 28 have 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 activation signal equal to 1. In that moment weights of RNN are updated and the next SQL statement is considered.

4 Training and Testing Data

We evaluated our system using data collected from PHP Nuke portal[10]. It is 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 and XSS attacks. A function of the portal related to executing 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. During operation of the portal we collected nearly 100000 SQL statements. It turned out that the medium length of a typical SQL query is quite big. To overcome this we used a compression mechanism, which consists in linking a keyword and data type into a new token. The length of encoded SQL statements decreased but it resulted in greater number of tokens, which requires more neurons in output layer of RNN. The set of all SQL queries was divided into 12 subsets, each containing SQL statements of different length. 80% of each data set was used for training and remaining data used for examining generalization. Teaching data are shifted one time forward in time.

Data with attacks are the same as reported in [1]. Below, we present SQL statements, which are the result of data typed in Web forms by an intruder. For a detailed description see [1]. The goal of the system is to recognize these queries as dangerous by predicting the next token, which is different than that, which was inserted by an intruder modifying a SQL query.

Attack 1

```
SELECT active, view FROM nuke_modules
WHERE title='Statistics';
UPDATE nuke_users SET user_password='<
new md5pass>' WHERE username='<user>';
--'
```

Attack 2

```
SELECT uname FROM nuke_session WHERE
uname='' OR username LIKE 'A%'; --'
```

Attack 3

```
SELECT user_password, user_id FROM
nuke_users WHERE username='' OR
user_password='<md5 password>' ;'
```

Attack 4

```
SELECT time FROM nuke_session WHERE
uname='\' OR user_password=\'<md5
password> \' ;'
```

Table 2. Value of parameters for training algorithm of Elman and Jordan Networks

index of data subset	length of data subset	η Elman	α Elman	η Jordan	α Jordan
1	2-4	0.2	0.2	0.2	0.0
2	5	0.3	0.2	0.6	0.1
3	6	0.2	0.05	0.5	0.05
4	7	0.2	0.05	0.4	0.1
5	8	0.1	0.05	0.3	0.05
6	9	0.2	0.0	0.3	0.1
7	10	0.1	0.05	0.2	0.0
8	11	0.1	0.05	0.2	0.2
9	12	0.1	0.05	0.2	0.05
10	13-14	0.1	0.0	0.1	0.1
11	15-16	0.05	0.1	0.1	0.1
12	17-20	0.05	0.1	0.1	0.05

Attack 5

```
INSERT INTO nuke_referer VALUES (NULL,
'"onclick="alert(document.domain);"')
```

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 layer, η that determines the extent of weights update. The table 2 presents the best values of parameters for each subset of SQL queries. For the Elman network all neurons in the hidden layer have sigmoidal activation function while all neurons in the output layer have *tanh* function. For the Jordan network *tanh* function was chosen for the hidden layer and sigmoidal function for the output layer. For such assignment of activation functions, the obtained RMS error was minimal. Ranges 2-4, 13-14, 15-16 and 17-20 of the data subset (see Table 2) means that these subsets include SQL queries of length between 2 and 4, 13 and 14, 15 and 16, 17 and 20. All remaining subsets contain fixed length statements. For each data subset we run RNNs 10 times for various initial value of weights, specifying 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 combination of settings of function activation for the hidden and output layer neurons. η and α parameters were taken in the range from 0.05 to 0.9. The table 2 summarizes this phase of experiments. For values presented in table 2 obtained error of RNN was minimal. The number of neurons in the hidden layer was also evaluated during experiments - for 58 neurons RMS error was minimal. One can see (see table 2) that in 75% cases η does not exceed 0.2 and α value in 87,5% of experiments is less than 0.2. In each epoch these values were constant.

The other approach is to set η value about 0.8 at the beginning and decrease it down to small value within next epochs. It results in moves of a point related to initial solution towards local minimum fast and then continuously slow down oscillating near local minimum.

In the second phase of the experimental study we trained 12 RNNs, one for each training data subset, using values from the first stage. Figure 4 shows how error of both networks changes through epochs. Input for both networks

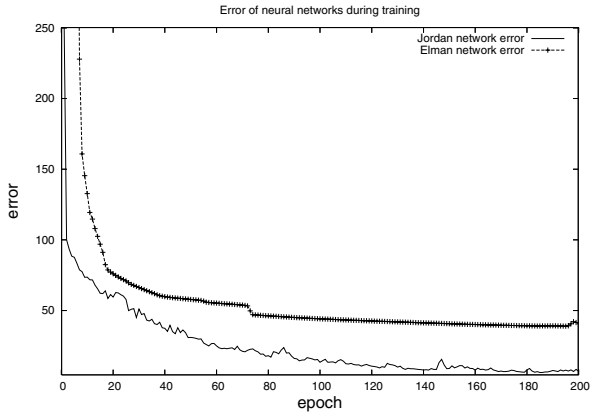

Figure 4. Neural networks training process for data subset with 10-length SQL queries

was a subset containing sequences of the length equals to 10. From the beginning of the training, the error of the Jordan network was much smaller than error of the Elman network. In the next a few epochs the error of both networks decreased quickly but the Jordan network error remained much smaller than the Elman network error. Figures 5 and 6 show how error of networks changes for all subsets of SQL queries. The figures also depict how well the networks are verified. Here, a statement is considered as well predicted if for all input vectors, all neurons in the output layer have values according to training data. All values presented in figures 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 the error of the Elman network. Despite of this for all data subsets percentage number of wrong predicted SQL queries for the Jordan network is less than the number of wrong predicted SQL statements for the Elman network. RMS (root mean square - eq.(4)) error of two RNNs varies for different data subsets and sometimes for longer queries it is less than error for shorter SQL statements. This is caused by different number of SQL statements included in each data subset. The number of input vectors depends on the number of SQL queries and the length of SQL query. What is important is a verification of a neural network. It states how good a net-

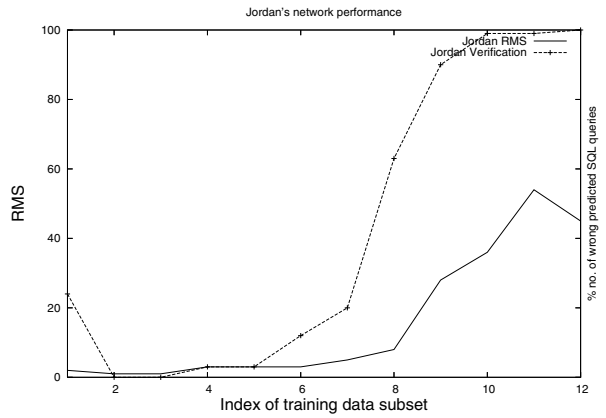

Figure 5. Error and number of wrong predicted SQL queries for each subset of data for Jordan's network

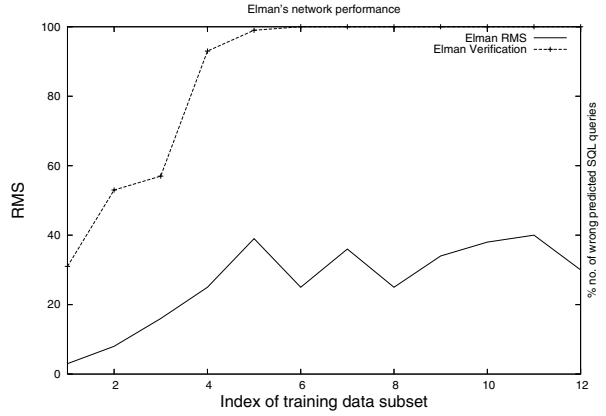

Figure 6. Error and number of wrong predicted SQL queries for each subset of data for Elman's network

work is trained. In the sense of the detecting of attacks, it means that the better verification, the less false alarms of a system. 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 from section 4. Each experiment was conducted using trained RNNs from the second stage. Figure 7 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 network output is different than the index 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

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

Figure 7. RNN output for an attack

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 8 shows RNN output for SQL statements derived from the training set and that, which was not present in the training set. The description of the figure 8 is similar to the description of the figure 7. 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. It is easy to see that the number of errors during verification

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

Figure 8. RNN output for known and unknown SQL statement

and generalization is much smaller than the number of errors when an attack is processed by RNN. Moreover, there is also more output vectors free of errors. Easily noticeable difference between an attack and normal activity allows us to re-evaluate obtained results presented in figures 5 and 6. Figure 8 presents typical outcomes for all trained RNNs and our training data. 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%. Apply-

Table 3. 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

ing these rules ensures that all attacks are detected by both RNN. The table 3 presents the percentage number of SQL statements wrongly predicted during verification and generalization if results were processed by the rules. For the most cases the Jordan network outperforms the Elman network. Only for 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 few legitimate statements, which were not in the training set were detected as attacks.

6 Conclusions

In the paper we have presented a new approach to detecting SQL-based attacks. The problem of detection was transformed to time series prediction problem and two RNNs were examined to show their potential use for such a class of attacks. It turned out that the Jordan network is easily trained by BPTT algorithm. Despite the fact that large architecture of RNN was used, that network is able to predict sequences of up to ten length with acceptable error margin. However, the Elman network characterizes greater error and because of large false alarms rate it is not supposed to detect attacks. The reason for why the Jordan network is much better than the other one is not obvious. We think that additional self feedback connection between neurons in the context layer can affect quality of a solution. The other, more important feature is the recurrent connection between different layers in both networks. The ability of prediction depends on the knowledge about gradient in the past. Because error of neurons in the output layer is greater than error of neurons in the hidden layer, more pressure is put on the hidden layer through the context layer in the Jordan network. Gradient ingredient in the Elman network is small and this causes that prediction is not based on the information about the past.

Deep analysis of the experimental results lead to the definition of rules used for distinguishing between an attack and legitimate statement. When these rules are applied, both networks are completely trained for all SQL queries included in the all training subsets . Accuracy of the results very strongly depends on the rules. The advisable part of experimental study is to apply defined rules to the other data set, which can confirm efficiency of the proposed approach to detecting SQL attacks.

This work shows that the Jordan network can be used to detect SQL-based attacks. To make proposed approach more robust, some additional features can be added to the attack representation. These features can be URL or IP address of a client connecting to a database. The lack of ability to detect attacks, that take advantage of longer SQL queries can be resolved by deploying other learning algorithms for RNN. In the future we are going to compare gradient-based algorithms with nature inspired algorithms, especially co-evolutionary genetic algorithms.

References

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

[2] C. Kruegel, G. Vigna, "Anomaly Detection of Web-based Attacks", Proceedings of the 10th ACM Conference on Computer and Communication Security (CCS '03), 2003, pp. 251-261

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

[4] K.M.C. Tan, K.S. Killourhy, R.A. Maxion, "Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits", In Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection, 2002, pp. 54-73

[5] I. Nunn, T. White, "The Application of Antigenic Search Techniques to Time Series Forecasting", In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005), USA

[6] M. Kendall, J. Ord, "Time Series", third edition, 1999

[7] D. Pollock, "A Handbook of Time-Series Analysis, Signal Processing and Dynamics", Academic Press, London, 1999

[8] T. Lin, B.G. Horne, P. Tino, C.L. Giles, "Learning long-term dependencies in NARX recurrent neural networks", IEEE Transactions on Neural Networks, 1996, pp. 1329

[9] P.R. Drake, K.A. Miller, "Improved Self-Feedback Gain in the Context Layer of a Modified Elman Neural Network", Mathematical and Computer Modelling of Dynamical Systems, 2002, pp. 307-311

[10] http://phpnuke.org/