**Jarosław SKARUZ** [1]

1 Siedlce University of Natural Sciences and Humanities
Faculty of Exact and Natural Sciences
Institute of Computer Science
ul. 3 Maja 54, 08-110 Siedlce, Poland

# Database security: combining neural networks and classification approach

DOI: 10.34739/si.2019.23.06

**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 the 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 a recurrent neural network (RNN) trained by backpropagation through time (BPTT) algorithm. Then, two coefficients of the rule are evaluated. The rule is used to interpret RNN output. In the testing phase RNN with the rule is examined against attacks and legal data to find out how evaluated rule affects efficiency of detecting attacks. All experiments were conducted on Jordan network. Experimental results show the relationship between the rule and a length of SQL queries.

**Keywords**. database, security, anomaly detection, neural networks

## 1. Introduction

Nowadays a lot of business applications are deployed in companies to support them with their business activity. These applications are often built with three layer manner: presentation, logical and data. Examples of data layer are files and databases containing data while the form of presentation layer can be desktop window or Web site presenting data derived from data layer and providing application functions.

The security concern related to business applications aims at ensuring integrity and confidentiality of data. Unfortunately, every a few weeks or days new security holes are discovered, which allow an attacker to break into application and steal data (http://www.securityfocus.com). As stated earlier all data can be stored in a database and logical layer is responsible to establish connection to a database, retrieve data and put them at presentation layer. To manage data in a database usually SQL statements are used. When a user executes an application function then SQL query is sent to a database and result of its execution is shown to the user. Possible security violations consist in not authorized access and modification of data in the database in case of poor implementation of an application. If a user can set values of some parameters of an SQL query, he can set malicious values, which leads to the change of the form of SQL query. While SQL is used to manage data in databases, its statements can be ones of sources of events for potential attacks.

In the literature there are some approaches to intrusion detection in Web applications. In [13] the authors developed anomaly-based system that learns the profiles of the normal database access performed by web-based application 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. While the obtained results are promising, the drawback of this work lies in the very small number of attacks used in the testing phase, which is statistically inappropriate.

Besides that work, there are some other works on detecting attacks on a Web server which constitutes a part of infrastructure for Web applications. Kruegel et al. [5] built a detection system that 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 system analyzes HTTP requests and builds data model based on the 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 [1] 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. Besides statistical methods there are some works, which takes advantage of neural networks. In [9] the authors use self organizing maps to detect anomalous network traffic. They successfully detected attacks performed on DNS and HTTP protocols. Ghosh et al. [3] used some machine learning

algorithms to detect misuse of privileged programs. Their system are able to detect user-to-root and program misuse classes of attacks with low false alarm rates. Tan et al. [11] 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 [15] present a survey which explores the various beliefs upon database forensics through different methodologies using forensic algorithms and tools for investigations. They refer to algorithms, forensic tamper detection of an audit log and present artifacts for database investigation.

In [16] the authors present two mixed techniques to secure the database. The first one is authentication followed by cryptography of database. The encryption algorithm is built on genetic algorithm, it is used to encrypt the database and validate the user's login id and password, it must verify the user and allow it to access the database. The decryption also has a login and password based on decryption algorithm and genetic algorithm.

Almutairi et al. [17] in their paper tackle various issues in database security such as the goals of the security measures, threats to database security and the process of database security maintenance.

A survey conducted to identify the issues and threats in database security, requirements of database security, and how encryption is used at different levels to provide the security is provided in [18]. In this study major security issues faced databases are identified and some encryption methods are discussed that can help to reduce the attacks risks and protect the sensitive data. It has been concluded that encryption provides confidentiality but give no assurance of integrity unless we use some digital signature or Hash function.

In this work we present a new approach to intrusion detection in applications. Rather than building profiles of normal behavior we focus on a sequence of tokens within SQL statements observed during normal use of an application. RNN is used to encode a stream of such SQL statements. In addition to this idea, we define classification rule and evaluate its coefficients, which increases efficiency of the intrusion detection system based on RNN.

The paper is organized as follows. The next section discusses SQL attacks. In section 3 we describe the architecture of Jordan network. Section 4 shows training and testing data used for experiments. Next, section 5 contains experimental results. Last section summarizes results.

## 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. 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)==0)
   echo "authorization failed";
else
   echo "authorization successful"
```

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 validated, 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. 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. Proposed approach

The way we detect intruders can be easily transformed to time series prediction problem. According to [7] 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:

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

where $S$ is any signal, which depends on a solved problem and $t$ is a current moment of 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 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 Q 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:

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

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 the 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 [4] and [8]. Besides statistical methods, RNNs have been widely used for that problem. In our study presented in this paper we use Jordan network.

## 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. Its value should be converged to zero. 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 are unknown now.

In comparison to feedforward neural networks, RNN have feedback connections which provide dynamics. When they process information, output neurons signal depends on input and output signal of neurons in the previous steps of training RNN. In the literature there are different architectures of RNN used for time series prediction problem [2], [6]. In this work we applied Jordan RNN.

### 3.2. RNN architecture

In comparison to feedforward neural network, Jordan network has context layer containing the same number of neurons as the output layer. Input signal for context layer neurons comes from the output layer. Moreover, Jordan network has an additional feedback connection in the context layer. Each recurrent connection has fixed weight equals to 1.0. Figure 1 presents architecture of Jordan network.

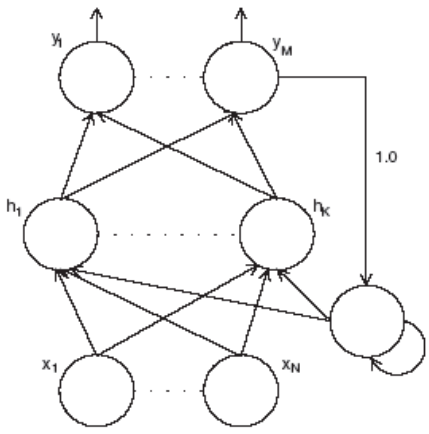


**Figure 1.** Jordan network

Network was trained by BPTT [12] and the related equations are presented below:

$$x(k) = [x_1(k),...,x_N(k), v_1(k-1),...,v_M(k-1)], \tag{4}$$

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

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

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

$$\delta_i^{(o)}(k) = [y_i(k) - d_i(k)] f'(g_i(k)), \delta_i^{(h)}(k) = f'(u_i(k)) \sum_{j=1}^{M} \delta_j^{(o)}(k) w_{ij}^{(2)}, \tag{8}$$

$$w_{ij}(k+1)^{(2)} = w_{ij}(k)^{(2)} + \eta \sum_{k=1}^{slq-length} [v_i(k) \delta_j^{(o)}(k)], \tag{9}$$

$$w_{ij}(k+1)^{(1)} = w_{ij}(k)^{(1)} + \eta \sum_{k=1}^{sql-length} [x_i(k) \delta_j^{(h)}(k)]. \tag{10}$$

In the equations (4)-(10), 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). Finally, in the last step values of weights are changed using formulas (9) for the output layer and (10) for the hidden layer.

### 3.3. Training

The training process of RNN is performed as follows. Subsequent 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 all tokens have 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, training data at the output layer are shifted so that RNN should predict the next token in the sequence of all tokens.

In this work, the following tokens are considered: keywords of SQL language, numbers and strings. We used the collection of SQL statements to define 36 distinct tokens. Each token has an index and the real number between 0.1 and 0.9, which reflects input coding of these statements according to table 1. The table 1 shows selected tokens, indexes and the coding real values.

**Table 1**. A part of a list of tokens, indexes and their coding values

| token | index | coding value |
|---|---|---|
| SELECT | 1 | 0.1222 |
| FROM | 2 | 0.1444 |
| … | … | … |
| … | … | … |
| UPDATE | 9 | 0.3 |
| … | … | … |
| number | 35 | 0.8777 |
| string | 36 | 0.9 |

The indexes are used for preparation of input data for neural networks. The index e.g. of a keyword *UPDATE* is 9. The index 2 points to a keyword *FROM*. The token with index 1 relates to *SELECT*. The number of neurons at the input layer is a parameter of the experiment. Network has 37 neurons in the output layer. 36 neurons correspond to each token but the neuron 37 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 equal to 0.1 or 0.9. If a neuron number $i$ in the output layer has small value then it means that the next processing token can not have index $i$. On the other hand, if output neuron number $i$ has value of 0.9, then the next token in a sequence should have index equal to $i$. Below, there is an example of a SQL query:

```
SELECT  name FROM users
```
(11)

At the beginning, the SQL statement is divided into tokens. The indexes of tokens are: 1, 36, 2 and 36 (see table 1). Here lets assume that the number of neurons at the input layer equals to 2. Then we have 3 subsequent pairs (1,36), (36,2), (2,36) of training data, each pair coded by real values in the input layer. Training data are presented in the figure 2.

The first two tokens that have appeared are 1 and 36. As the consequence, in the first step of training, the output signal of two neurons in the input layer are 0.1222 and 0.9. The next token in a sequence has index equal to 2. This token should be predicted by the network in the

result of the input pair (1,36). It means that only neuron number 2 at the output layer should have the high value. In the next step of training, 2nd and 3rd tokens are presented to the network, which means that output of neurons is set according to tokens 36 and 2.

0.1222 0.9

0.9 0.1444,

0.1444 0.9.

**Figure 2.** Input data for the network

Fourth token should be predicted so neuron number 36 in the output of RNN should have the high value. Finally, input data are 0.1444 and 0.9. Only neuron number 37 at the output layer should have high value, which means that this training vector is the last within considered SQL query. In that moment weights of RNN are updated and the next SQL statement is considered.

Training the network in such a way ensures that it will posses prediction capability. While network output depends on the previous input vectors, processing the set of tokens related to the next SQL query can not be dependent on tokens of the previous SQL statement. Thus after updating weights output signal of all neurons in the context layer is set to 0.

## 4. Training and Testing Data

All experiments were conducted using synthetic data collected from a SQL statements generator. The generator takes randomly a keyword from selected subset of SQL keywords, data types and mathematical operators to build a valid SQL query. Since the generator was developed on the basis of the grammar of SQL language, each generated SQL query is correct. We generated 3000000 SQL statements. Next, the identical statements were deleted.

Finally, our data set contained thousands of free of attack SQL queries. The set of all SQL queries was divided into 20 subsets, each containing SQL statements of different length, from 10 to 29 tokens. Data with attacks were produced in the similar way to that without attacks. Using available knowledge about SQL attacks, we defined their characteristic parts.

Next, these parts of SQL queries were inserted randomly to the generated query in such a way that it provides grammatical correctness of these new statements. Queries in each subset were divided into three parts: for training RNN, evaluating coefficients of a classification rule and testing RNN with the rule. Each of the part contains 500 SQL statements.

## 5. Experimental Results

In the first phase of experimental study we evaluated the best parameters of RNN and learning algorithm. For the following values of parameters the error of the networks was minimal. For the Jordan network *tanh* function was chosen for the hidden layer and sigmoidal function for the output layer.

The number of neurons in the hidden layer equals to 38 neurons. In all cases $\eta$ (training coefficient) is set to 0.2. While output signal of a neuron in the output layer can be only high or low, for each input data there is a binary vector as the output of the network. If a token is well predicted by the network, it means that there is no error in the vector.

Otherwise, if at any position within the vector there is 0 rather than 1 or 1 rather than 0, it means that the network predicted wrong token. Experimental results presented in [10] show that there is a great difference in terms of the number of errors and the number of vectors containing any error for cases, in which attack and legal activity were executed. An example of the network output in terms of the number of errors is shown in the table 2.

**Table 2**. The number of errors at the output of RNN for legal and malicious queries with 10 tokens.

| Index of data vector | Legal SQL query | Malicious SQL query | |
|---|---|---|---|
| | | Used during training | Unknown for the RNN |
| 1 | 3 | 1 | 3 |
| 2 | 1 | 0 | 1 |
| 3 | 2 | 1 | 0 |
| 4 | 3 | 2 | 2 |
| 5 | 2 | 0 | 0 |
| 6 | 3 | 0 | 1 |
| 7 | 2 | 0 | 0 |
| 8 | 2 | 1 | 3 |
| 9 | 0 | 0 | 0 |

To distinguish between an attack and a legitimate SQL statement we defined the classification rule: an attack occurred if the average number of errors for each output vector is not less than $\alpha$ and $\beta$ is greater than the number of output vectors that include any error (Skaruz et al., 2007). Mathematical form of the rule is presented in eq. 12.

$$\begin{cases} \text{number of errors} > \alpha, \\ \text{number of vectors with errors} > \beta. \end{cases} \tag{12}$$

The problem with that rule is that assumed values of coefficients can not be used generally. They depend on the length of SQL query. The objective of this research is to show the

relationship between the values of the rule coefficients and the length of SQL query. Also, we present in what extent the number of false alarms depend on the length of SQL query. Moreover, we show how the number of SQL statements used for setting the coefficients of the rule affects efficiency of intrusion detection.

### 5.1. Experiment description

The scenario of the experiment is presented below:

```
Initialization: divide set of SQL queries on 20 subsets
(each one contains different length queries)
FOR history=2 to 5
     FOR each subset
          train a network
               FOR k=1 to 10
                    FOR i=1 to 9
                         choose randomly i/10 queries
                         examine the network against chosen
queries
                    END
               END
          evaluate coefficients of the rule
          test the networks with the rule
     END
END
```

**Figure 3.** Scenario of the experiment.

In the experiment there are three subset of SQL queries for each network. SQL statements from the first subset were used for training the network. The second subset was used for the rule coefficients calculation and the last one for testing both the network and the classification rule.

For each data subset we had one network, which was trained using data from a distinct subset. The whole experiment was repeated for different length of historical data. After the network was trained using data with attacks, it was examined against attacks and legitimate SQL queries. This procedure was performed using data from the subset, which was not used during training the network. First, 10 % of data was selected randomly and put to the network and the network output was recorded.

This step was repeated 9 times increasing amount of data by 10%. This procedure was repeated ten times to calculate average results. Taking different number of SQL queries to calculate the rule coefficients allows to find the optimal size of data subset used for this purpose. The greater subset the more accurate coefficient are evaluated. On the other hand large number of SQL statements affects computational complexity.

The purpose is to find the lowest number of SQL queries, which allows to calculate good coefficients. Next, based on the network output for legal and illegal SQL statements, we calculated two coefficients of the rule so that the average of false positive and false negative alarms is minimal. In the last step of he experiment the RNN with the rule was tested using data from the third distinct data subset.

## 5.2. Results

### The number of SQL queries used for coefficients calculation vs. false alarm rate

The objective of this part of experimental study was to calculate optimal number of SQL queries used for calculation of the rule coefficients. The number should be as small as possible allowing for small false alarm rate at the same time. In this section we show results for the lowest, highest and middle length of SQL queries.

Figure 4, 5 and 6 present the values of coefficients of the rule for different number of SQL queries used for training the network and for calculating the coefficients. The length of historical data equals to 5.

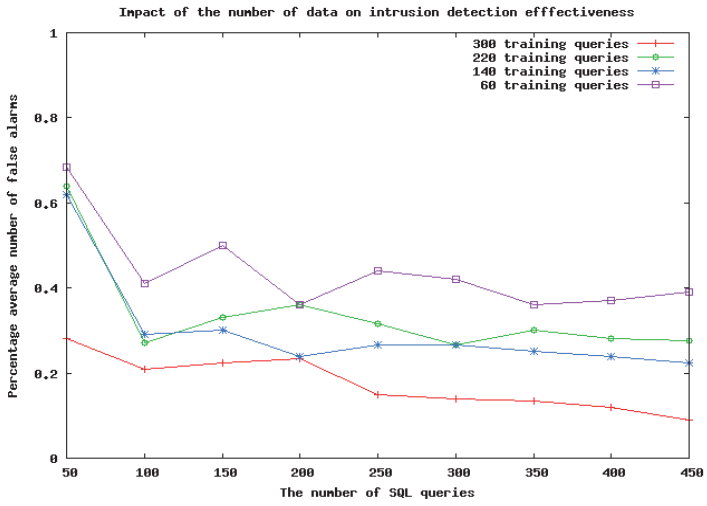


**Figure 4.** Relationship between the number of SQL queries used for training, the number of data used for coefficients calculation and average of false alarms. Length of SQL queries equals to 10.

For the shortest SQL queries we obtained very good results, over 99% of SQL queries were classified successfully. The best results are received from the network trained with the greatest number of SQL statements. However, it is worth pointing out that the difference between the networks trained with various number of data are only slight and one may wants to save training time by choosing only few SQL queries. The impact of the number of SQL statements used for coefficients calculation on detection rate is also little and it is better to chose 50 queries.

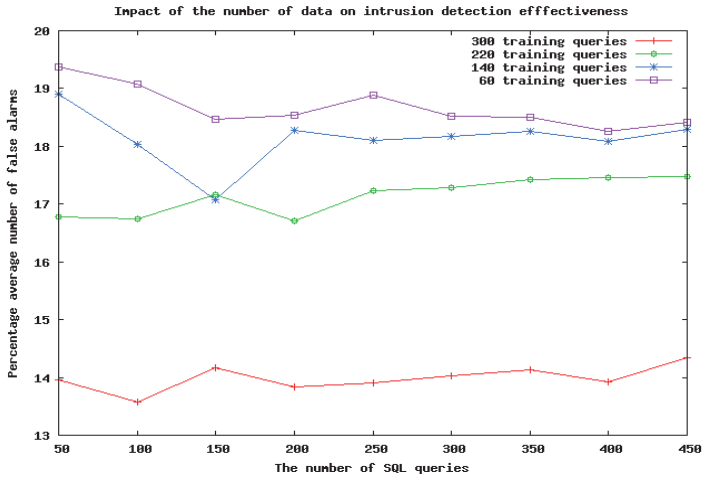


**Figure 5.** Relationship between the number of SQL queries used for training, the number of data used for coefficients calculation and average of false alarms. Length of SQL queries equals to 20.

The results on figure 5 are appropriate for the length of SQL queries equals to 20. The false alarm rate increased by about 14% in the best case in comparison to the results on SQL statements with 10 tokens. There is no relation between false alarm rate and the number of data used for coefficients calculation. Decreasing computational complexity by choosing fewer number of data used for training the network costs about 4% of false alarm rate.

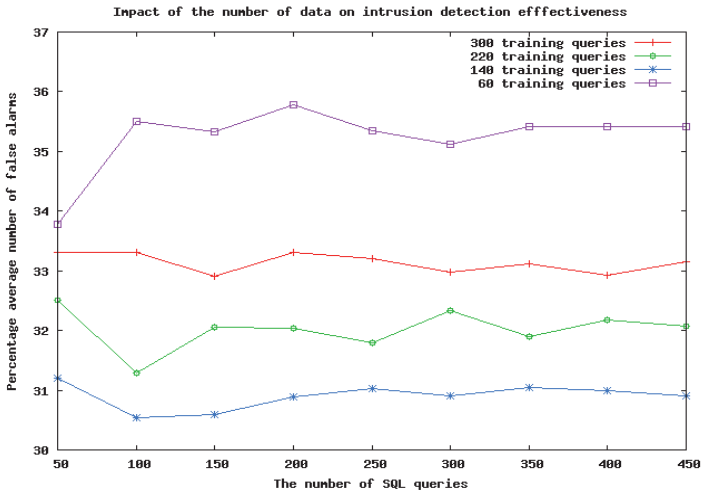


**Figure 6.** Relationship between the number of SQL queries used for training, the number of data used for coefficients calculation and average of false alarms. Length of SQL queries equals to 29.

Figure 6 presents experiment results on 29 tokens SQL queries. Similarly to the previous two figures there is no relation between efficiency and the number of data used for coefficients evaluation. Increasing the size of training set probably will not improve results. The longer SQL queries the more difficult is to train the network. We think that the limitation of this approach to intrusion detection is the length of SQL statement equals about 30.

**The length of historical data vs. false alarm rate**

In this part of experimental study the objective was to check how the length of historical data affect false alarm rate. Figure 7 shows histogram of false positive rate, false negative rate and their average.

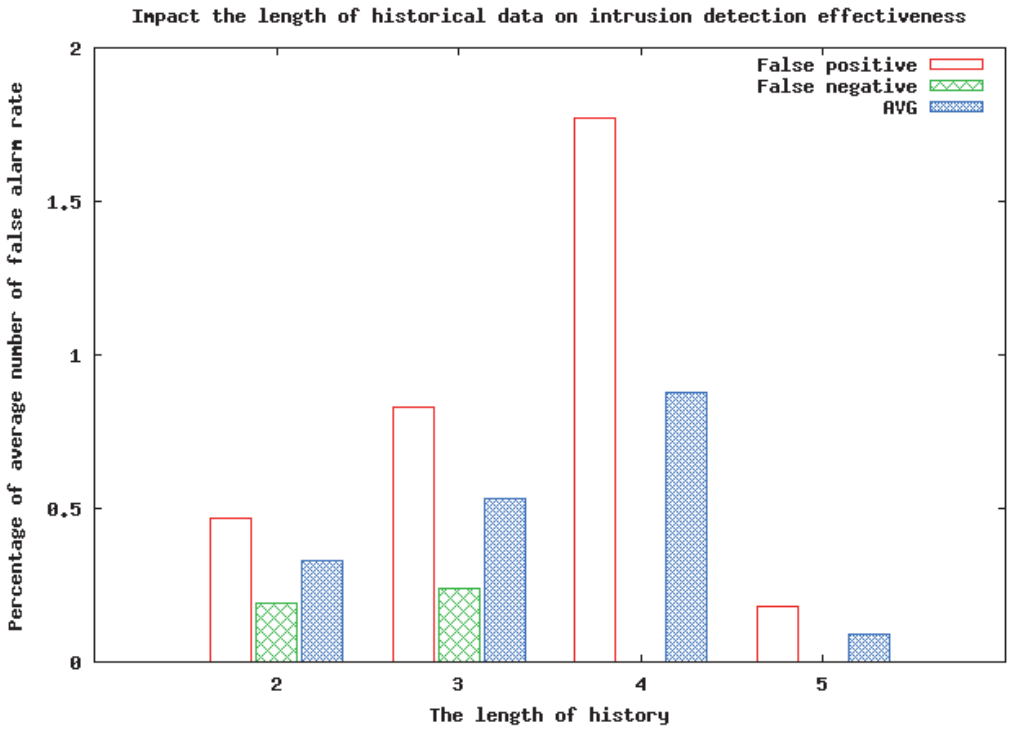


**Figure 7.** False alarms rate for various length of historical data. Length of SQL queries equals to 10.

Average false alarm rate increases when the length of history increases from 2 to 5 tokens. For length equals to 5, average false alarms rate decreases much. Since different sequences of data are input for network in each case (see section 3.3), it is difficult to say why this relation exists. Figure 8 relates to SQL queries with 20 tokens. Here, the lowest average of false alarm rate is obtained for history equals to 4. Figure 9 presents results for SQL statements constituted from 29 tokens. In that case the length of historical data does not affects effectiveness of intrusion detection rate.

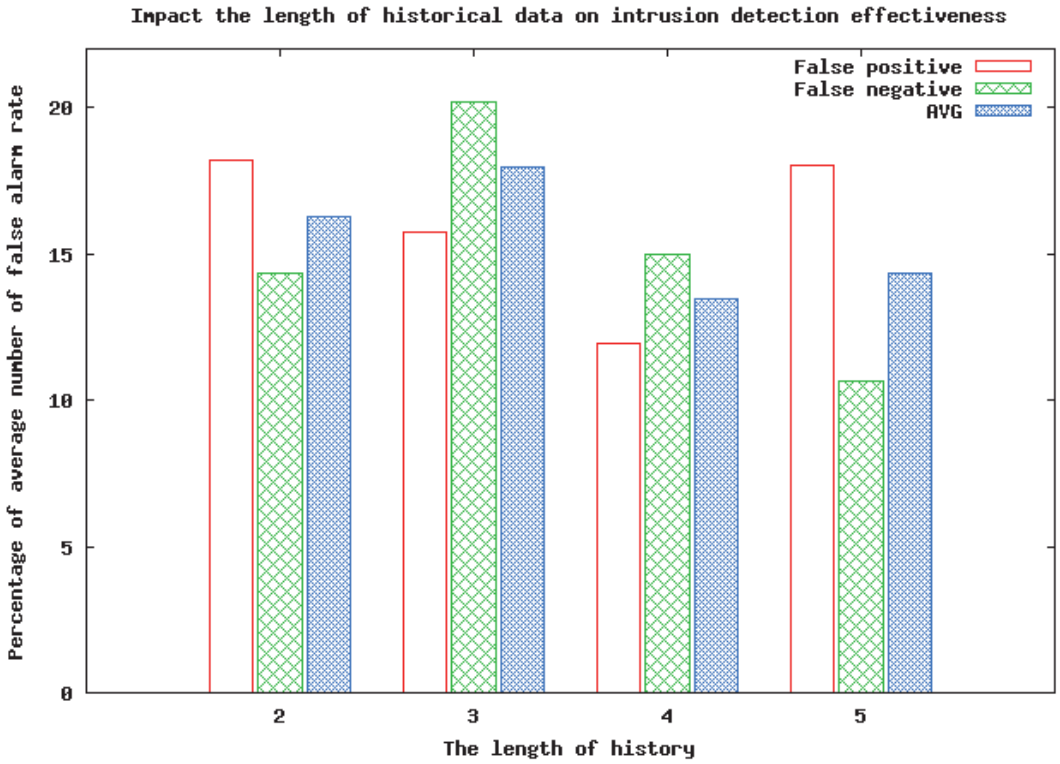


**Figure 8.** False alarms rate for various length of historical data. Length of SQL queries equals to 20.

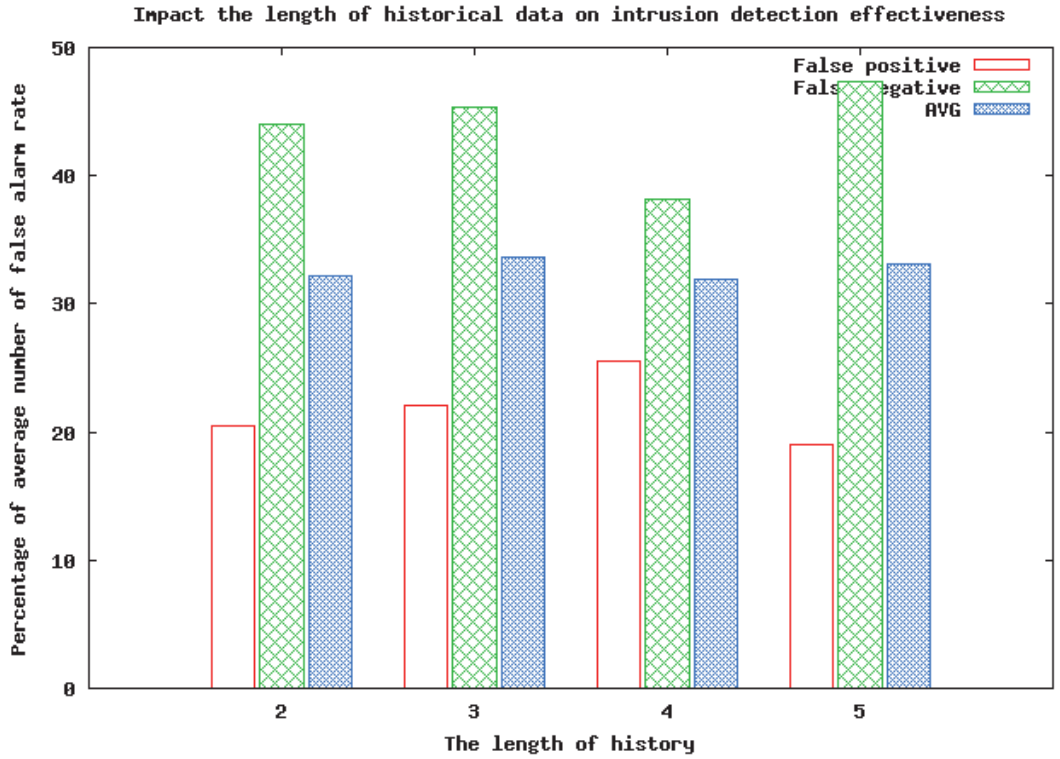


**Figure 9.** False alarms rate for various length of historical data. Length of SQL queries equals to 29.

### The length of SQL statements vs. false alarm rate

Figure 10 shows false alarm rate for each considered in this work length of SQL queries.

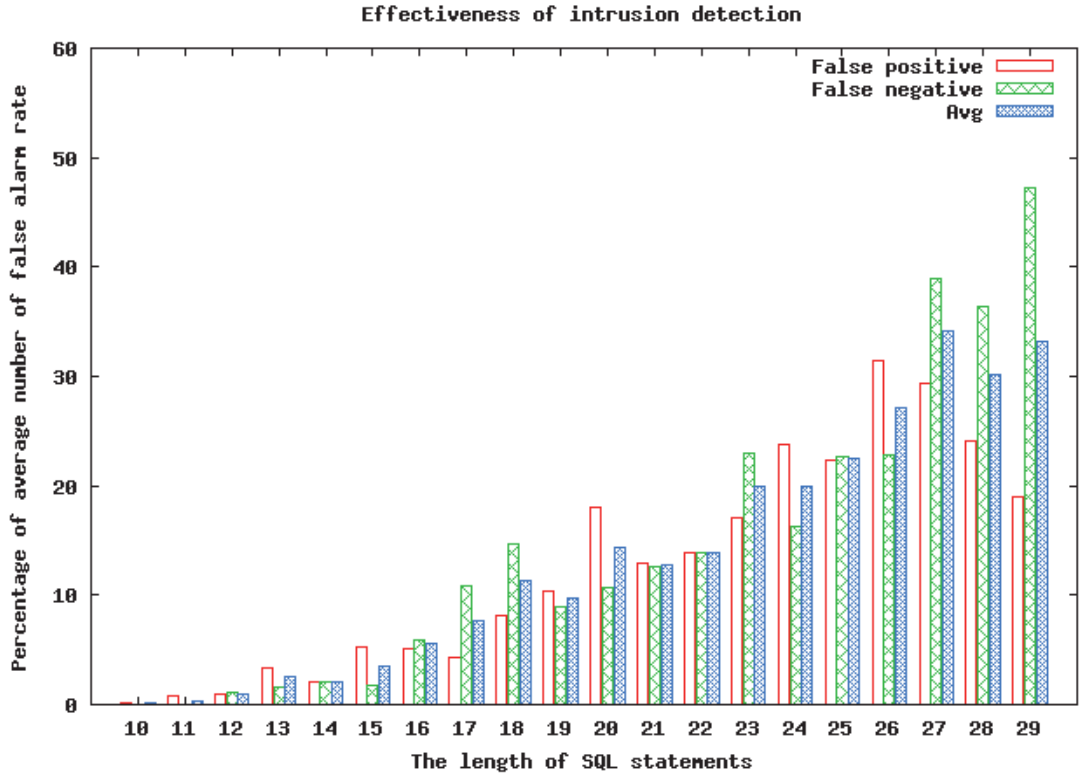


**Figure 10.** Effectiveness of intrusion detection for various length of SQL statements.

To obtain the results presented in figure 10 neural network was trained with 300 SQL queries, 450 queries was used for the rule coefficient calculation and 5 historical data were used. The best results (0.09% of average of false alarm rate) we have obtained for SQL queries with 10 tokens. Detecting SQL attacks based on SQL queries with up to 19 tokens is possible with the average of false alarm below 10%. Statements built with 20-24 tokens are classified correctly in 80% of cases. The efficiency of this approach to intrusion detection decreases for the longer SQL statements and varies from 22% to 33 % of false alarms.

### The length of SQL statements vs. the rule coefficients

In this work it is crucial to calculate the coefficients of the classification rule. Figure 11 shows the relation between coefficients and the length of SQL queries. The results presented in figure 11 were obtained from the network with 5 input neurons trained with 300 SQL queries. It can be seen that the dependence between coefficients and the length of SQL queries is similar

to linear. The efficiency of RNN is limited by the number of training data, especially if large neural net architecture is used. Moreover, the quality of trained network depends on the length of SQL query. The longer SQL statement, the more difficult is to train it. For such cases, output vectors of RNN include more errors and there is little difference in the output network when attacks and legal SQL queries are presented to the network. That is the reason for which greater values of coefficients must be used to discriminate between attacks and legal activity.

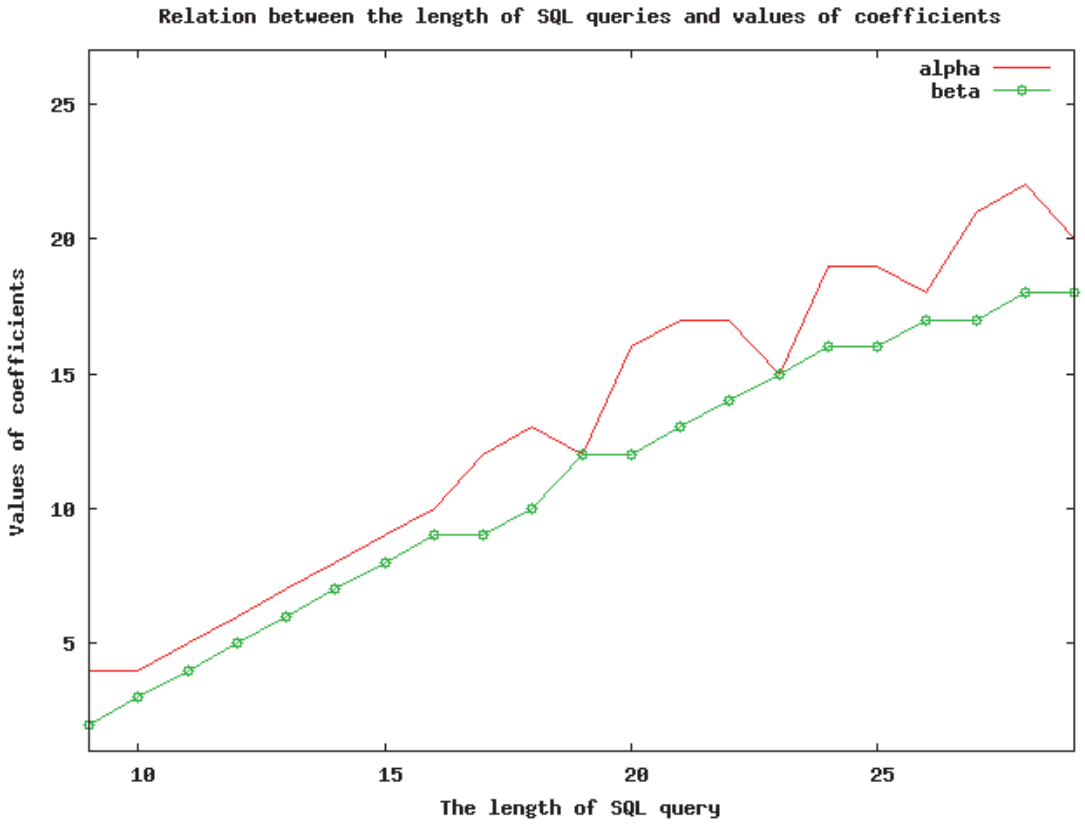


**Figure 11.** Values of the rule coefficients for different length of SQL queries.

### The length of historical data vs. the rule coefficients

Figure 12 and 13 show how the length of historical data affects values of both coefficients. It can be seen that increasing the length of history leads to decreasing values of both coefficients. The reason for that relation is that the length of history affects the number of training vectors for RNN. Let $h$ means the length of history and $t$ is a number of tokens within a SQL query. Then the number of training vectors $l$ can be calculated according equation (13).

$$l = t - h + 1. \tag{13}$$

If $h$ is increased then the number of training vectors decreases. Ideal trained RNN means that for each data input the number of errors in the output layer of RNN equals to 0. It is natural

that if the number of training vectors decreases then the number of errors in the output of RNN decreases.

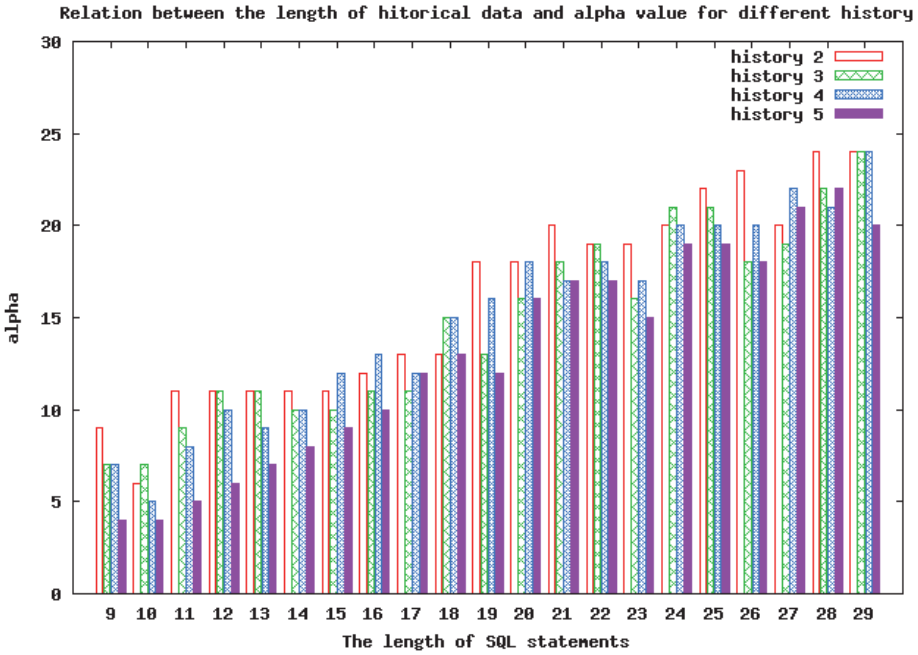


**Figure 12.** Alpha coefficient value for different length of SQL queries and length of historical data.

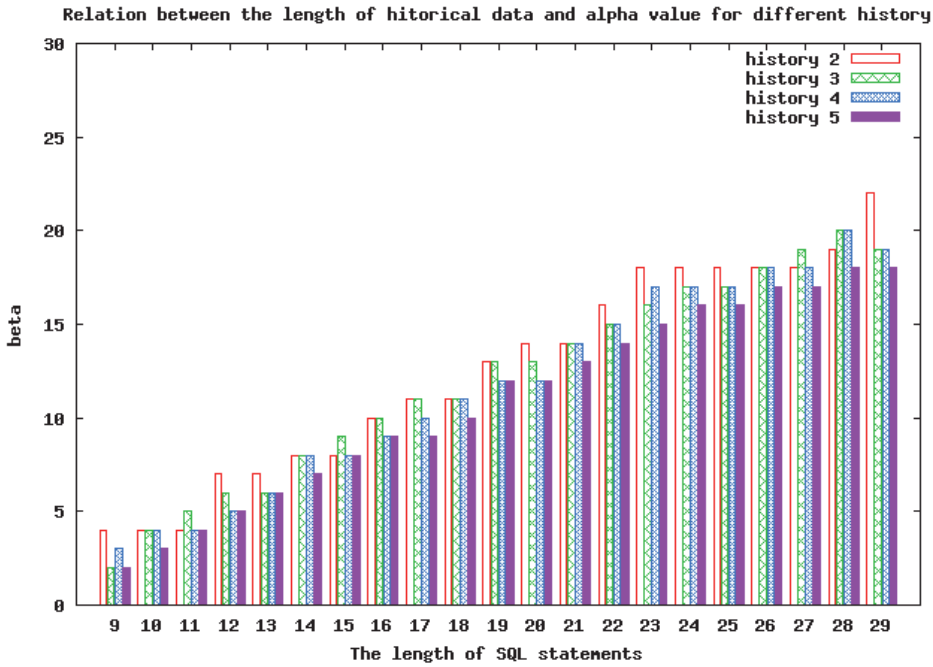


**Figure 13.** Beta coefficient value for different length of SQL queries and length of historical data.

## 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 Jordan network was examined to show its potential use for such a class of attacks. Despite the fact that large architecture of RNN was used, the network is able to predict sequences of the length up to twenty nine tokens with acceptable error margin.

Finally, the classification rule was defined and validated experimentally. The relationship between the coefficients of the rule and the length of SQL query was established. We also showed how the number of SQL queries used for setting the coefficients affects the number of false alarms. The results presented in this work are helpful especially when this approach to intrusion detection is deployed in real environment, where there is no any clue about the optimal number of data, which allow to define the rule without decreasing the efficiency of intrusion detection.

## Acknowledgments

I would hereby like to thank Prof. dr hab. Franciszek Seredynski for his help and contribution during the writing of my PhD dissertation. This paper is based on the author's doctoral thesis.

## References

[1] 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.

[2] 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, pp. 307-311.

[3] Ghosh A. K., Michael C., Schatz M.: A Real-Time Intrusion Detection System Based on Learning Program Behaviour, Recent Advances in Intrusion Detection, Springer, LNCS, 2000, pp. 93-109

[4] Kendall M., Ord J.: Time Series, third edition, 1999.

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

[6] 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, pp. 1329.

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

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

[9] Ramadas M., OsterMann S., Tjaden B.: Detecting Anomalous Network Traffic with Self-organizaing Maps, Recent Advances in Intrusion Detection, Springer, LNCS, 2003, pp. 36-55

[10] Skaruz J., Seredyński F., Bouvry P.: Tracing SQL Attacks via Neural Networks, Parallel Processing and Applied Mathematics, Springer, LNCS, 2007.

[11] Tan K.M.C., Killourhy K.S., Maxion R.A.: 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.

[12] Werbos P. J.: Backpropagation Through Time: What It Does and How to Do It, in Proceedings of IEEE, Vol. 78, 1990, pp. 1550-1560

[13] 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 (DIMVA), Austria, 2005.

[14] Internet source: http://securityfocus.com.

[15] Khanuja H., Adane D. S.: Database Security Threats and Challenges in Database Forensic: A Survey, International Conference on Advancements in Information Technology With workshop of ICBMG, 2011

[16] Sabareesan M., Gobinathan N.: Network Database Security Issues and Defense, International Journal of Engineering Research and Applications, Vol. 3, Issue 1, 2013, pp. 1748-1752

[17] Almutairi A.H., Alruwaili A. H.: Security in Database Systems, Global Journal of Computer Science and Technology Network, Web & Security, Vol. 12, Issue 17, 2012

[18] Basharat I., Azam F., Muzaffar A. W.: Database Security and Encryption: A Survey Study, International Journal of Computer Applications, Vol. 47, No.12, 2012