A Learning-based Neural Network Model for the Detection and Classification of SQL Injection Attacks

*Naghmeh Moradpoor Sheykhkanloo
School of Computing (SoC), Edinburgh Napier University
Edinburgh, United Kingdom
n.moradpoor@napier.ac.uk

***Abstract*— Structured Query Language injection (SQLi) attack is a code injection technique where hackers inject SQL commands into a database via a vulnerable web application. Injected SQL commands can modify the back-end SQL database and thus compromise the security of a web application. In the previous publications, the author has proposed a Neural Network (NN)-based model for detections and classifications of the SQLi attacks. The proposed model was built from three elements: 1) a Uniform Resource Locator (URL) generator, 2) a URL classifier, and 3) a NN model. The proposed model was successful to: 1) detect each generated URL as either a benign URL or a malicious, and 2) identify the type of SQLi attack for each malicious URL. The published results proved the effectiveness of the proposal. In this paper, the author re-evaluates the performance of the proposal through two scenarios using controversial data sets. The results of the experiments are presented in order to demonstrate the effectiveness of the proposed model in terms of accuracy, true-positive rate as well as false-positive rate.**

Keywords*— *Intrusion Detection, SQL injection attacks, machine learning, Artificial Intelligence, Neural Networks, Web Attacks, Databases

I. INTRODUCTION

SQL is a programming language designed for handling data in a Relational Database Management System (RDBMS) [17]. SQLi attack is a technology weakness that comes from dynamic script language such as PHP: Hypertext Processor (PHP), Active Server Pages (ASP), Java Server pages (JSP) and Common Gateway Interface (CGI). It takes advantages of inappropriate and/or poor coding of web applications that allows hackers to inject malformed SQL commands in order to gain un-authorised access to data resides in the related back-end database.

For any organisation, data contains important and confidential information that can be related to them, their customers, and their business partners. This information can range from personal or less sensitive information such as: first name and last name to more sensitive information such as: username, password, pin code, and credit card information. If inputs from a user-side are not properly sanitised, a hacker can generate crafted SQL commands and can inject them into a database in order to pass say a login barrier and see what exists behind it. This leads to sever damages on a given database such as: disclosing, modifying, and/or removing data or in a worse-case scenario wiping the entire database. Therefore, it is important for any organisation to protect their databases in order to prevent any loss to themselves, their customers, and their business partners.

SQLi attack has been ranked as the most harmful danger, A1-Injections, in top 10 security threats for web applications in Open Web Application Security Project (OWASP) [13]. An A1-Injection attack includes injection flaws such as: SQL, OS, and LDAP injections. This occurs when unsafe and/or untrusted data is sent to an interpreter as part of a command or query tricking it into executing unintended commands or accessing data without a proper authorisation.

CIA triad, which stands for: Confidentiality, Integrity, and Availability, is a well-known security model that can be used to develop a security policy for any organisation. If a given database is attacked, CIA elements can be violated. For instance, the data in the database can be revealed to unauthorised users, which is a failure in Confidentiality element of the CIA triad. The data can be altered, which is a failure in Integrity element of the CIA triad. In a worst-case scenario, the data can be completely wiped out from the database which is a failure in Availability element of the CIA triad.

In the previous work [14], the author proposed a NN-based model for SQLi attack detections which built from three elements: a URL generator, a URL classifier, and a NN model. Addressing the published results, the previous proposal was successful to detect the malicious URLs from the benign URLs. The author then extended the proposal to a pattern recognition NN-based model for the detection and classification of the SQLi attacks [15]. Addressing the published results, the proposed model was successful to not only detect the malicious URLs from the benign URLs, but also classify the malicious URLs into the popular SQLi attack categories. Finally, in the most recent work [16], the author stress tested the previous proposals where the model demonstrated a good performance in terms of accuracy. In this paper, the author further investigates the performance of the previous proposal [14-16] by implementing two different test beds and scenarios. This includes employing different sets of data for the developed NN-based model in order to demonstrate the effectiveness of the proposed technique.

The remainder of this paper is organised as follows. In Sections II, the author reviews the related work for the

TABLE 1 SQL INJECTION ATTACK TYPES, SIGNATURES, AND PREVENTIONS [15]

Type of SQLi attack	Signature	Prevention on a user side	Prevention on a database side
Tautologies (Type1)	‘, OR, =, like, select	-Strictly validating user inputs	-Blocking queries containing tautological condition WHERE clauses
Illegal/logically incorrect queries (Type2)	invalid conversions (CONVERT (TYPE)), incorrect logics, AND, ORDERBY, ‘	-Strictly validating user inputs	-Stopping and/or sanitising generated error messages (e.g. logical errors, type errors and syntax errors) from a given database
Piggy-backed query (Type3)	;	-Strictly validating user inputs	-Avoiding multiple statement executions on a database by scanning all queries for delimiter “;”
Union queries (Type4)	UNION, UNION SELECT	-Strictly validating user inputs	-Blocking multiple query executions in a single statement
Stored procedures (Type5)	;, Stored procedure keywords (SHUTDOWN, exec, xp_cmdshell(), sp_execwebtask())	-Strictly validating user inputs -Giving proper roles and privileges to stored procedures being used in a web application	-Using a low privileged account to run a database -Executing stored procedures with a safe interface
Inference SQLi attack (Type6)	;, AND, IF ELSE, WAITFOR	-Strictly validating user inputs	-Carefully crafting error messages return from databases -Patching/hardening databases
Alternate encoding (Type7)	;, exec (), Char (), ASCII (), BIN (), HEX (), UNHEX (), BASE64 (), DEC (), ROT13 ()	-Strictly validating user inputs, for instance prohibiting any usage of meta-characters e.g. “Char ()”	-Treating all meta-characters as normal characters

detections and preventions of the SQLi attacks. The author’s previous proposal [14-16] and the related implementations are discussed in Sections III & IV, respectively. Sections V and VI include two different scenarios along with the related results using three sets of data. This is followed by conclusions of the work in Section VII, acknowledgments, and references.

II. RELATED WORK FOR SQL INJECTION ATTACKS

In this section, existing work related to the SQLi attack detection and prevention techniques are addressed as follows.

Authors in paper [2] proposed an algorithm based on Support Vector Machine (SVM) in order to detect and classify SQLi attacks. Addressing their captured results, their proposed algorithm presented 96.47% accuracy for SQLi attack detections.

Authors in paper [3] proposed a static analysis tool for checking Java Database Connectivity (JDBC) to verify the correctness of dynamically generated SQL queries. Addressing their captured results, their proposed JDBC checker flags potential errors or verify their absence in dynamically generated SQL queries with low false positive rate.

In order to detect SQLi attacks, authors in paper [4] proposed a query tokenisation algorithm where QueryParser was employed. They have assumed that there is no way someone can perform a SQLi attack without inserting space, single quote, and/or double dashes in a query. Therefore, they designed two arrays: one for the original queries and one for the injected queries where each element is a token obtained from the related query. At the end, they obtained the length of each resulting array and compared them. Henceforth, if two arrays have different length there is a SQLi attack.

In order to prevent SQLi attacks, authors in paper [5] proposed a Random4 encryption algorithm based on randomisation where user input values, e.g. usernames and passwords are converted into cipher text using a

lookup table. The encrypted key can then be stored in a database and compared with the user inputs received during the login time. In order to evaluate the performance of their proposed algorithm they employed techniques such as: brute force attack and dictionary attacks in order to crack the related keys stored in the database. They also compared their proposal with existing algorithms such as: AMNESIA [8], SQL rand [10], SQL DOM [9], WAVES [11], and SQL check [12] in terms of encoding, detection, and prevention.

Authors in paper [6] proposed a Service Based SQL Injection Detection (SBSQLID) algorithm which is positioned between a given application server and the related database. SBSQLID includes three elements: input validator, query analyser, and error service. The input validator retrieves user inputs from a web application and passes them into a set of injection characters for pattern matching. Thus, if pattern matching returns false, the user will be able to work with the web application otherwise he/she will be disallowed. After validating the user inputs, they will be passed into the query analyser for syntactic and semantic structure verifications. The last element of their proposal is an error service where any error messages produced by the database server will be generalised and then sent back to the application server. This is done in order to stop attackers for receiving any Meta-data information from a back-end database.

Authors in paper [7] proposed a translation and validation-based solution for SQLi attacks, TransSQL, where SQL requests are automatically translated to Lightweight Directory Access Protocol (LDAP)-equivalent requests. SQL and LDAP-equivalent queries are then executed on SQL database and LDAP database, respectively. At the end, TransSQL checks the difference in responses from both databases in order to detect and then block any SQLi attempts.

Authors in paper [8] proposed AMNESIA stands for Analysis and Monitoring for NEutralising SQL Injection

Attacks. AMNESIA was proposed in order to detect and prevent SQLi attacks by combining static analysis and runtime monitoring. The static analysis was used in order to analyse the entire codes in a web application and automatically build a model for legitimate queries that a given web application can generate. The runtime monitoring was then employed in order to monitor all dynamically generated queries and check whether they are different from the static generated model. At the end, queries that violate the static model were classified as SQLi attacks and prevented from any access to the database.

Authors in paper [9] proposed SQL Domain Object Model (DOM) for compile time checking instead of runtime checking of dynamic SQL statements. Using SQL DOM, application developers are able to build dynamic SQL statements through manipulation of objects, which are strongly typed to the database, without the need for string manipulations.

In order to detect and prevent SQLi attacks, authors in paper [10] proposed a randomised SQL query language, SQLrand, where the standard keywords in SQL were manipulated by attaching a randomised and a hard to guess integer to them. To achieve probability and security, their proposed SQLrand includes a proxy server that sits between a client and a database in order to receive randomised SQL quires from a client and de-randomised them before passing them to the back-end database. Addressing their captured results, the latency overhead that imposed on each query by using SQLrand is negligible thus it does not sacrifice the performance.

Authors in paper [11] proposed a Web Application Vulnerability and Error Scanner (WAVES) as a security assessment tool in order to identify poor coding practices that render web applications vulnerable to attacks such as SQLi and cross-site scripting attacks. A number of software testing techniques such as: dynamic analysis, black-box testing, fault injection, and behaviour monitoring was described and took into account in their implementations. At the end, WAVES was compared with other vulnerability scanner tools where it has been proven as a feasible platform for assessing web application security.

Authors in paper [12] proposed SQLCHECK as a runtime checking algorithm to prevent SQLi attacks. Their proposed algorithm was evaluated in real-world web applications with real-world attack data as inputs where SQLCHECK produces no false negative and no false positive. Addressing their captured results, SQLCHECK also has low run-time overhead and can be applied straightforwardly to web applications written in different languages.

After studying the exiting work related to the SQLi attack detection and prevention techniques, the author has noticed a huge lack of employing Artificial Intelligence (AI) in this filed. AI has been successfully used in a wide range of fields including: medical diagnosis, stock trading, robot control, law, remote sensing, scientific discovery, and toys. AI studies how to create computers and computer software that are capable of intelligent behaviour just like human beings. Artificial neural Networks (NNs) is one of the popular AI algorithms which has been employed in various fields in order to perform complex functions that are difficult for conventional computers or human beings. For instance: pattern recognition, identification, classification, speech, vision, and control systems. This motivates us to bring the SQLi attack detection and prevention problem into the AI filed and particularly into the application of the NN. Our ultimate research objective is to provide a NN-based Intrusion Detection and Prevention (ID&IP) tool that can be easily extended from SQL-IDS to any application level attacks e.g. Deny of Service (DoS), drive-by downloads, phishing email, and Man-In-The-Middle (MIMT) attacks.

In author's previous work [14], she proposed a NN model for the detection of SQLi attacks. Her proposed technique was successful to classify a given URL as either a benign URL or a malicious URL. This has been done by taking into account the popular SQLi attack keywords and URL patterns. The author then improved this initial proposal in [15] by adding another level of intelligence where her proposed model was successful to not only detect the malicious URLs from the benign URLs but also to detect the type of SQLi attacks, Table 1 [15], for the malicious URLs and classify them accordingly. In the author's most recent work [16], our previous work from [14] and [15] has been tested in order to demonstrate the effectiveness of our proposed technique. In this paper, we re-evaluate the performance of our previous proposal in terms of accuracy, true-positive rate, and false positive-rate through two scenarios. This includes using different sets of URLs for the URL generator that leads to different classifications made by the URL classifier and different decisions made by the NN model. The author's previous proposal from [14-16] is discussed in the next section.

III. PROPSOED MODEL

In this section, we explain three elements of our previous proposal [14-16] which includes: the URL generator, the URL classifier, and the NN model as follows, Figure 1[15].

A. The URL Generator

The URL generator has two components: "Benign URLs" and "Malicious URLs". The "Benign URLs" includes the real URL addresses that exit in the world and have/have not SQLi attack signature(s). These URLs have been captured from [18]. The Google search engine [19] has been employed in order to find the URL addresses which are benign but have SQLi attack signature(s). The "Malicious URLs" includes the malevolent and harmful URL addresses that have SQLi attack signature(s). These URLs have been generated by adding the SQLi attack signature(s) to the most popular URL addresses in the world [18] using the PHP scripting language [20].

B. The URL Classifier

The URL classifier is responsible for: 1) identifying a given URL as a benign URL or as a malicious URL, and 2) detecting the type of SQLi attack for the malicious URLs. Basically, the URL classifier deals with the URL addresses which are presented/generated by the URL generator. The author has mathematically defined the URL classifier's functionalities as follows.

Let a URL characteristic r_i is defined by a random variable R_i as follows:

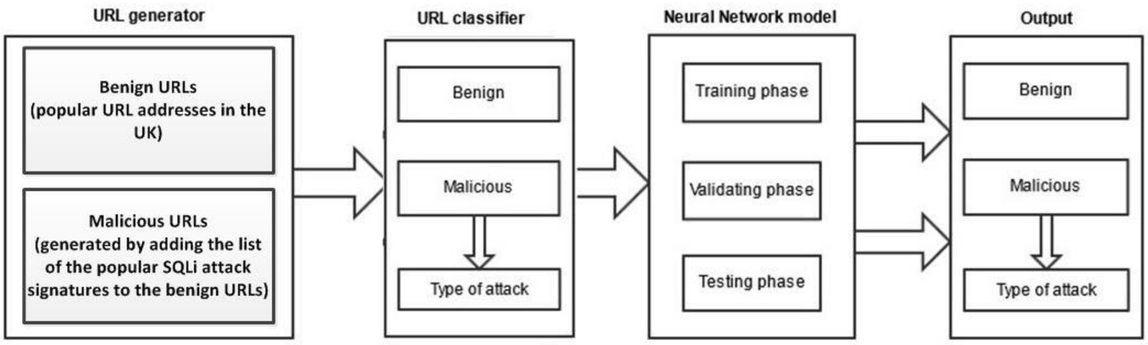


Figure 1. Components of the proposed neural network-based model [15]

R_i=

$$\begin{cases} 1, \text{if discovered by the SQLi signature detectors} \\ 0, \text{if not discovered by the SQLi signature detector} \end{cases}$$

Let *C* be a random variable indicating the URL's class which can be either malicious or benign:

$C\epsilon$ {malicious, benign}

Each URL (malicious/benign) is assigned with a vector defined by $r^- = (r_1, r_2, , \ldots, r_n)$ with r_i being the result of the i-th random variable R_i.

Let a malicious URL characteristic t_i is defined by a random variable T_i:

T_i=

$$\begin{cases} 1, \text{if discovered by the SQLi attack type detectors} \\ 0, \text{if not discovered by the SQLi atatck type detector} \end{cases}$$

Let *D* be a random variable representing the type of the malicious URLs, which can be: "Tautologies", "Illegal/logically incorrect queries", "Piggy-backed query", "Union queries", "Stored procedures", "Inference SQLi attack", or "Alternate encoding", Table 1[15]:

$D\epsilon$ {Tautologies, Illegal/logically incorrect queries, Piggy-backed query, Union queries, stored procedures, Inference SQLi attack, Alternate encoding}

Each malicious URL is assigned with a vector defined by$t^- = (t_1, t_2, , \ldots, t_n)$, with t_i being the result of the t-th random variable R_i.

C. *The Neural Network (NN) Model*

The NN model deals with the URL addresses that have already been classified into either benign or malicious. It also has knowledge about the type of SQLi attack for malicious URLs. The NN model receives this information from the URL classifier and takes it into account for three phases of: training, validating and testing with distribution rates of 70%, 15%, and 15%, all respectively.

The NN model includes *x* inputs and *y* outputs which are connected through *n* hidden layers/neurons via directed arrows, Figure 2 [15]. Each directed arrow can have different value. The value called connection weigh or simply weight.

In order to learn the weights, the author has employed a popular NN-based algorithm called backpropagation which is an abbreviation for backward propagation algorithm. The algorithm starts with a set of inputs (a set of random weights) and a set of desire outputs. By using the inputs and the random weights, the author first let the network calculate some outputs. Obviously, as the weights are selected randomly, there will be differences between the calculated outputs and the desire outputs. Thus, the calculated outputs will be compared with the desire outputs and the differences will be measured. The differences between these two sets called network errors. Now, the network knows about the errors, it tries to adjust the weights in order to produce the outputs which are closer to the desire outputs and thus have smaller errors.

The author has mathematically defined the backpropagation algorithm as follows.

Let the weight for the i-th node defined by a random variable $W_{j,i}$ (left side of the arrow below); where $W_{j,i}$ (right side of the arrow below) is the node's old weight, α is the learning rate, a_j is the node's input value, and Δ_i is the network error. Therefore, the new weight is calculated and then adjusted as follows.

$$W_{j,i} \leftarrow W_{j,i} + \alpha \, x \, a_j \, x \, \Delta_i$$

The error for the i-th node (Δ_i) is calculated as follows.

$$\Delta_i = (T_i - O_i) \, x \, g' \, (\textstyle\sum_j W_{j,i} \, a_j)$$

For simplicity, the author puts together the three components of the NN model in two groups as follows.

1) *Training Elements*

The *Training Elements* include three components as follows.

- "Input Matrix": this matrix contains all the URLs (malicious and benign) that the NN model uses in training stage. These URLs have been generated/presented by the URL generator.
- "Target Matrix": this matrix includes all the decisions (malicious or benign) for all the URLs as well as the type of SQLi attacks for malicious URLs. These decisions cover each URL stored in the 'Input' matrix.

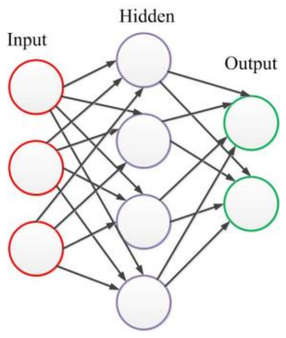


Figure 2. An artificial neural network model [15]

- "Fitness Network": this is the NN model with *n* layers with *x* inputs and *y* outputs where the data from 'Input' and 'Target' matrixes are used for training, validating, and testing, respectively.

2) *Validating/Testing Elements*

The *Validating/Testing Elements* of the NN model include two components as follows.

- "Sample Matrix": this matrix contains sample data from the "Input Matrix". The trained NN model uses the data in the "Sample Matrix" as inputs during the validation phase.
- "Output Matrix": this matrix contains output data for the data in the "Sample Matrix". The trained NN model predicts the output values for the "Sample Matrix" and stores them in the "Output Matrix".

The implementation of the proposed NN model is discussed in the next section.

IV. IMPLEMENTATIONS

The implementation of the proposed NN model is discussed as follow.

A. *The URL Generator*

The two elements of the URL generator, Benign URLs and Malicious URLs, are implemented as follows.

1) *The "Benign URLs"*

The "Benign URLs" holds the real URL addresses and includes two separate lists: List1 and List2. List1 contains a list of the real URL addresses which are benign with absolutely no SQLi attack signature. List2 includes a list of the real URLs which are also benign but with SQLi attack signature(s). The author has considered 6,250 real URL addresses for List1 and List2. Therefore, the benign URL addresses come to 12,500 in total. As an example URL for List1, consider the Google's URL address in the UK [19]. This is a benign/real URL address with absolutely no SQLi attack signature(s). As an example URL for List2, consider the European Union's URL address in Wikipedia [21]. This is also a benign/real URL address but has a SQLi attack keyword of "union". As a result of this, the proposed NN model can falsely detect the European Union's URL as a malicious URL and thus classify it as a "Union queries" SQLi attack. This is called false positive. Furthermore, the author divides List2 into five sub-lists: List2.1, List2.2, List 2.3, List2.4, and List 2.5, where 20% (1,250 URLs), 40% (2,500 URLs), 60% (3,750 URLs), 80% (5,000 URLs), and 100% (6,250 URLs) of the URLs has this issue. For instance, in List 2.1, only 10% of the benign URLs have SQLi attack signature(s) whereas in List 2.5 100% of the URLs has this issue. This comes to 6,250 URLs for List2 in total. The reason to employ different data sets in this paper is to re-evaluate the effectiveness and the performance of the author's previous proposal [14-16] in terms of: accuracy, false-positive, and false-negative rates.

2) *The "Malicious URLs"*

In this paper, the author generated the malicious URLs by simply adding SQLi attack signature(s), Table 1[15], to the benign URLs using a PHP script. For instance, addressing the "Piggy-backed query" SQLi attack signatures, a generated malicious URL can be a benign URL that has delimiter ";". Likewise, adding ";, SHUTDOWN, exec" signatures to a benign URL, generates a malicious URL which can be identified as a "Stored procedures" SQLi attack. The total malicious URLs come to 12,500.

B. *The URL Classifier*

The URL classifier is accountable for: 1) identifying whether a given URL is a benign URL or a malicious URL, and 2) identifying the type of SQLi attack for a given malicious URL. Given that 1 represents true/malicious and 0 represents false/benign in this paper, the author encoded these two tasks using string of logic for each URL. This has been done by allocating two vectors: r^{-} and t^{-} to each URL where r^{-} has 32 features ($\mathrm{r}^{-} = (\mathrm{r}_0, \mathrm{r}_1, , \dots, \mathrm{r}_{31})$) and t^{-} has 8 features ($\mathrm{t}^{-} = (\mathrm{t}_0, \mathrm{t}_1, , \dots, \mathrm{t}_7)$). For instance, if a URL includes: " ; ", "AND", "IF", "ELSE" and "WAITFOR" keywords, it will be classified as a malicious URL with "00000000000101000000111000000000" value for r^{-} where: r_{11} represents " AND ", r_{13} represents ";", r_{20} represents "IF", r_{21} represents "ELSE", and r_{22} represents "WAITFOR". Moreover, given that this is an "Inference SQLi attack", which is the attack type6 in this paper, the value for t^{-} vector is "00000010". The r^{-} and t^{-} components are shown in Table 2 [15] and Table 3 [15], respectively.

C. *The Neural Network (NN) Model*

The NN model employs 70% of the total URLs for "Training phase", 15% for "Validating phase", and 10% for "Testing phase". Given that the author has implemented two scenarios to re-evaluate the effectiveness of our previous proposal, and that each scenario has different NN configurations, the author identifies the NN elements independently for each scenario. Therefore, the two implemented scenarios (SCENARIO1 and SCENARIO2) are designed as follows.

V. SCENARIO1: DETECTION AND STRESS TEST

In SCENARIO1, which is called DETECTION AND STRESS TEST, the author focuses on responsibility number one for the NN model: identifying the malicious URLs from the benign URLs. This scenario has been taken into account in order to re-evaluate the performance of the previous proposal [14-16] in terms of: accuracy, true-positive rate, and false positive-rate with different and controversial data sets. The data sets are controversial as there is a great chance for the NN model to falsely detect the malicious URLs as benign and vice versa. For each category of benign and malicious URLs, the author has taken into account the equal number of 12,500 URL addresses with the following specifications.

The URL addresses in benign category are grouped in two lists: List1 and List2 each list carries 6,250 URLs, Table 4. In order to make the decision making (benign/malicious) harder, List2 is further divided into five sub-lists: List2.1, List2.2, List 2.3, List 2.4, and List 2.5, where 20% (1,250 URLs), 40% (2,500 URLs), 60% (3,750 URLs), 80% (5,000 URLs), and 100% (6,250 URLs) of the URLs has the issue of being benign but having SQLi attack signature(s). The total number of the URLs in each sub-list is 6,250. The URL addresses in malicious category are the malicious URL addresses which carries the SQLi attack signature(s). The total number of the URLs in this category is 12,500 with distribution rate of: 1,800 URLs for Type1 to Type 6 and 1,700 URL addresses for Type 7 SQLi attack, Table 4.

TABLE 2. ASSIGNED VECTORS FOR SQLI ATTACK SIGNATURES [15]

Vectors	**SQLi attack signatures**
r_0	‘
r_1	or
r_2	=
r_3	like
r_4	select
r_5	convert
r_6	int
r_7	char
r_8	varchar
r_9	nvarchar
r_{10}	incorrect logics
r_{11}	and
r_{12}	orderby
r_{13}	;
r_{14}	union
r_{15}	union select
r_{16}	shutdown
r_{17}	exec
r_{18}	xp_cmdshell()
r_{19}	sp_execwebtask()
r_{20}	if
r_{21}	else
r_{22}	waitfor
r_{23}	--
r_{24}	ascii()
r_{25}	bin()
r_{26}	hex()
r_{27}	unhex()
r_{28}	base64()
r_{29}	dec()
r_{30}	rot13()
r_{31}	*

In Scenario1, all the malicious and benign URLs, which are 12,500 URLs in each category that gives 25,000 URLs in total, are classified by the URL classifier to either benign or malicious. This task has been done by developing: 1) one PHP script to convert all 25,000 URL addresses to strings of logics (zeros and ones), and 2) one PHP script to identify each URL either as benign or malicious. In both scripts, 1 represents true/malicious and 0 represents false/benign. This gives us two matrices that act as inputs to our proposed NN model: Input Matrix and Target Matrix. They are named as Training Elements.

TABLE 3. ASSIGNED VECTORS FOR SQLI ATTACK TYPE [15]

Vectors	**SQLi attack type**
t_0	Benign
t_1	Tautologies
t_2	Illegal/logically incorrect queries
t_3	Piggy-backed query
t_4	Union queries
t_5	Stored procedures
t_6	Inference SQLi attack
t_7	Alternate encoding

TABLE 4. BENIGN AND MALICIOUS URLS FOR SCENARIO 1 & 2

Benign URLs						**Malicious URLs**						
Total: 12,500 URLs						**Total:** 12,500 URLs						
List1	**List2**					**Type 1**	**Type 2**	**Type 3**	**Type 4**	**Type 5**	**Type 6**	**Type 7**
Total: 6,250 URLs	**Total:** 6,250 URLs											
	List 2.1	**List 2.2**	**List 2.3**	**List 2.4**	**List 2.5**	500	2,500	500	1,000	1,500	2,500	4,000
	1,250 + 5,000	2,500 + 3,750	3,750 + 2,500	5,000 + 1,250	6,250							
Total: 25,000 URLs												

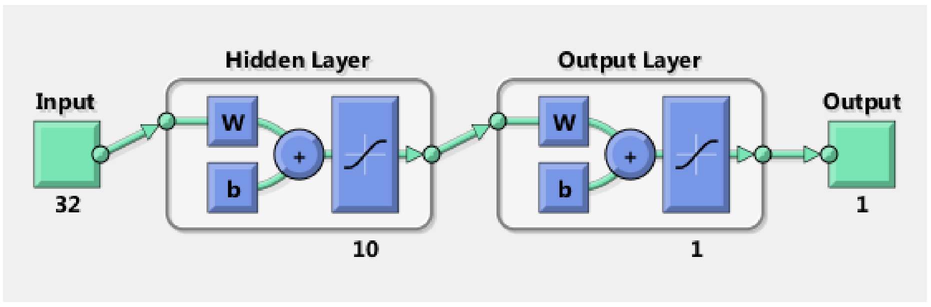


Figure 3. Network Architecture for the Neural Network component of the proposed model (Scenario1)

There are two other matrices that act as output to the proposed NN model: Sample Matrix and Output Matrix. They are named as Testing/Validating Elements. For simplicity, the author has merged the validating and testing elements together with the following specifications for Scanrio1.

1) *Scenario1: Training Elements*

- "Input Matrix": this matrix is a logical *25,000 x 32* matrix where the URLs are represented in strings of logics, *0s* (benign) and *1s* (malicious). It is a matrix with *25,000* rows and *32* columns. *25,000* represent the number of the URLs in this scenario, which is *12,500* for benign URLs and *12,500* for malicious URLs precisely. The *32* represents the size of the assigned vectors to the SQLi attack.
- "Target Matrix": this matrix is a logical *25,000 x 1* matrix where the URLs are represented in string of logics, *0s* (benign) and *1s* (malicious). It is a matrix with 25,000 rows and 1 column. 25,000 represent the number of the URLs while 1 represents the size of the assigned decision vector to each URL which is either benign (0) or malicious (1).
- Fitness network: this is our NN model with *10* hidden nodes or *10* layers/neurons where 70%, 15%, and 15% of the data from 'Input' and 'Target' matrices are used for training, validating, and testing, all respectively.

2) *Scenario1:Validating/Testing Elements*

- "Sample Matrix": this matrix is a logical *n x 32* matrix contains *n* sample data from the "Input Matrix".
- "Output Matrix": this matrix is a logical *n x 1* matrix contains output data for the URLs represented in "Sample Matrix". The trained NN model predicts the output value, in terms of a URL being benign or malicious, for each URL in the "Sample Matrix". These predictions will be stored in the "Output Matrix".

The whole data set, which includes 25,000 URLs and 25,000 decisions (malicious/benign), will be used in order to train (by 70% of data), validate (by 15% of data), and test (by 15% of data). The Scenario has been implemented in MATLAB [22]. The NN model has 10 hidden layers, 32 input features, 1 output layer, and 1 output features, Figure 3. For Scenario1, the results have been captured, represented, and analyzed in two groups of: confusion matrix and Receiver Operating Characteristic as follows.

The confusion matrices for all three phases of training, validating, and testing are depicted in Figure 4 to Figure 8. Each figure shows different results with different sets of date. For instance, Figure 4 shows the output results when the input data includes 12,500 benign URLs (6,250 URLs from List 1 and 6,250 URLs from List 2.1) and 12,500 malicious URLs while Figure 8 shows the output results when the input data comes from List1, List 2.5, and malicious URLs. This gives the total of 25,000 URLs for each experiment.

As it is depicted in Figure 4 to Figure 8 and based on the implementations, there are two output classes and two target classes: class 0 which represents benign URLs and class 1 which represents malicious URLs. For each class, the number of the correct responses is shown in a green square and the number of the incorrect responses is shown in red square. The grey square illustrates the percentages of the accuracies (upper numbers) and inaccuracies (bottom numbers) for output and target classes. The blue square shows the overall percentages of the accuracies (upper numbers) and inaccuracies (bottom numbers) for each experiment.

For instance in Figure 4, three phases of training, validating and testing receive 25,000 URLs (11,749 + 501 + 751 + 11,999). This includes 12,250 benign URLs (11,749 + 501), which are in class 0, and 12,250 malicious URLs (751 + 11,999) which are in class 1. The former is 49.0% (47.0% + 2.0%) and the latter is 51.0% (3.0% + 48.0%) of the total URLs used in this phase. For the benign URLs, which are in class 0, the percentage of correct responses is 95.9% while the percentage of the incorrect responses is 4.1%. These percentages are 94.1% and 5.9% for malicious URLs, which are in class 1, all respectively. This gives the total percentage of 95.0% accuracies and 5.0% of inaccuracies.

Moving from Figure 4 to Figure 8, as it was expected, the author realises that the total percentage of accuracies are reduced and the total percentage of inaccuracies are increased for each set of experiment gradually by 5.0%. For instance, in Figure 5, in which the author employs URLs from List 2.2 where 40.0% of the URLs (total of 2,500) have the issue of being benign but having SQLi attack signature(s), the percentage of accuracies is 90.0% and inaccuracies is 10.0%. However, these percentages reach 75.0% and 25.0% correspondingly in Figure 8, in which the

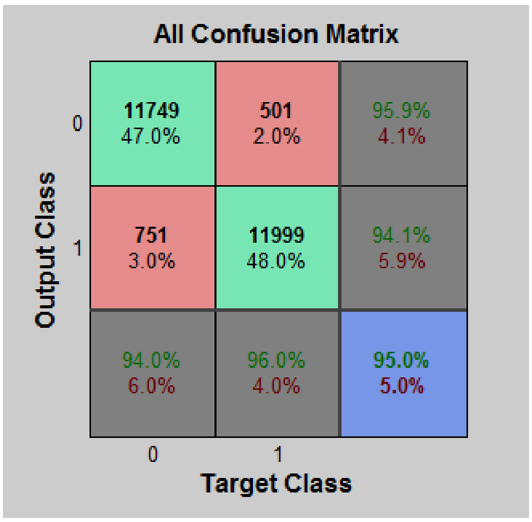


Figure 4. Confusion matrix for three phases in Scenario1 (URLs from: List1 & List 2.1 & Malicious URLS)

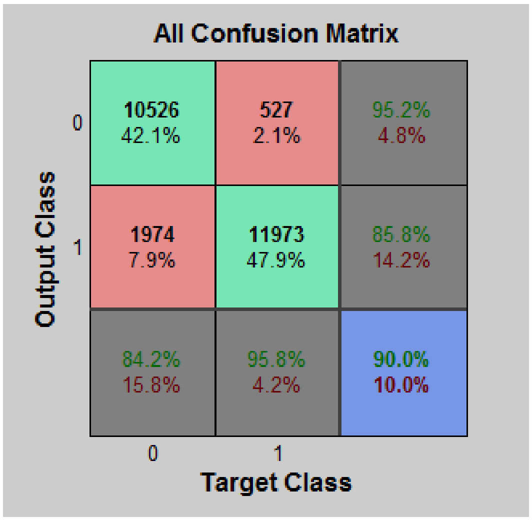


Figure 5. Confusion matrix for three phases in Scenario1 (URLs from: List1 & List 2.2 & Malicious URLS)

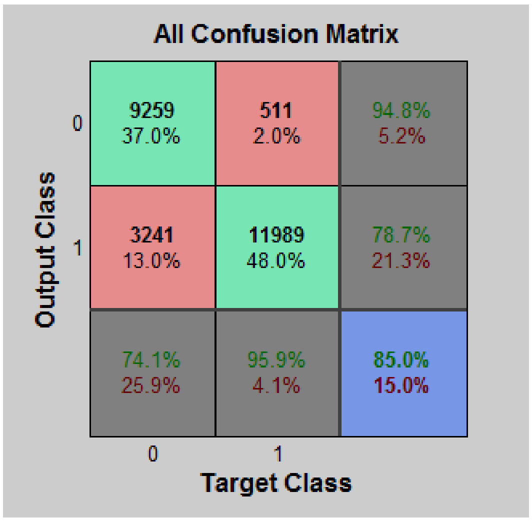


Figure 6. Confusion matrix for three phases in Scenario1 (URLs from: List1 & List 2.3 & Malicious URLS)

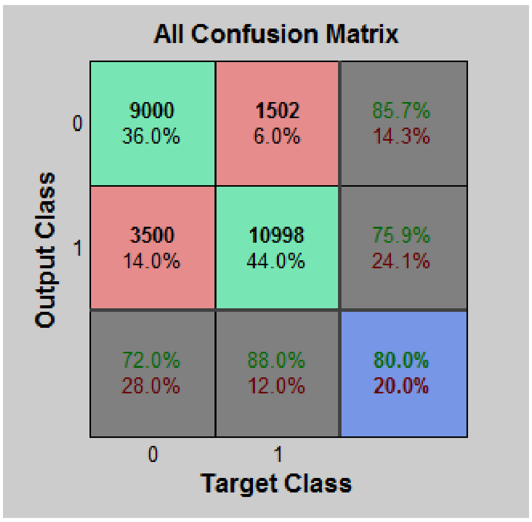


Figure 7. Confusion matrix for three phases in Scenario1 (URLs from: List1 & List 2.4 & Malicious URLS)

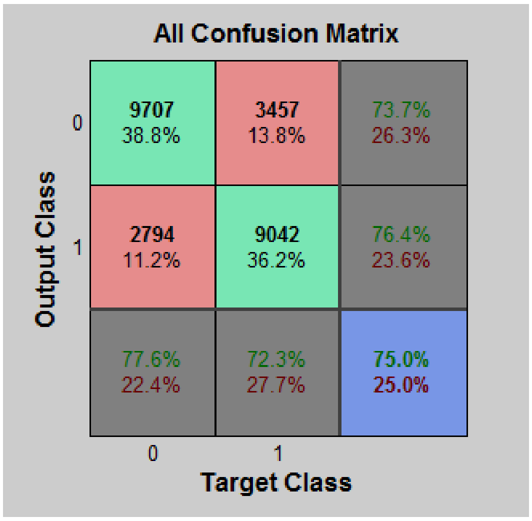


Figure 8. Confusion matrix for three phases in Scenario1 (URLs from: List1 & List 2.5 & Malicious URLS)

author employs URLs from List 2.5 where 100.0% of URLs (total of 6,250 URLs) have the issue of being benign but having the SQLi attack signature(s).

The network performance for all three phases (training, validating, and testing) and for all five sets of experiments in Scenario1 are captured in Figure 9 to Figure 11. This has been measured in terms of mean squared error and has been shown in log scale. The mean squared error is the difference between output and target. Thus the lower values are better and zero means there is no error in the network. Moving from Figure 9 to Figure 11, the author notices that, as it was expected, the network performance is gradually reduced by almost 0.03ms from 0.02ms in Figure 9 to 0.13ms in Figure 11. This is because of the different data set that the author employed in each set of experiments which makes it more difficult for the network to reach its best performance when gradually got from Figure 9 to Figure 11. This means the network takes longer to get its best performance. For instance, the network performance is better in Figure 10 (i.e.

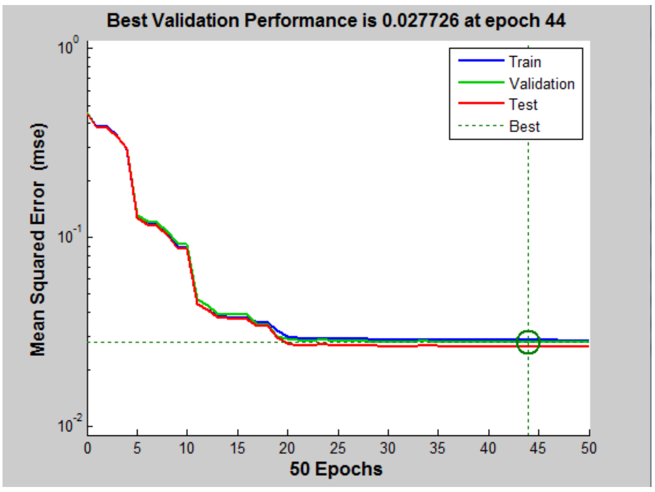


Figure 9. Network performance for Scenario1 (URLs from: List1 & List 2.1 & Malicious URLS)

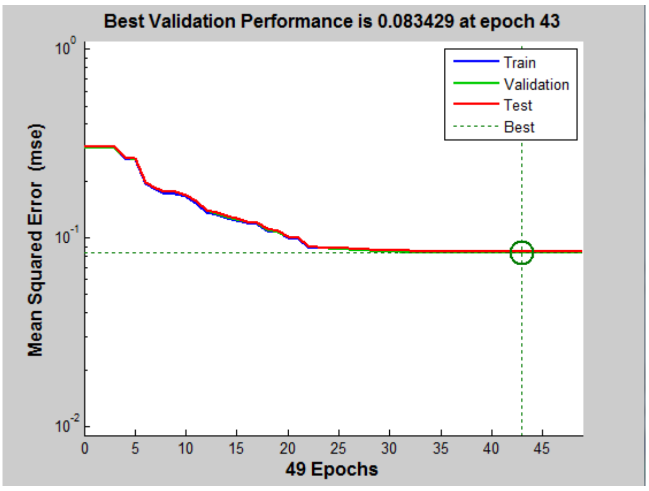


Figure 10. Network performance for Scenario1 (URLs from: List1 & List 2.3 & Malicious URLS)

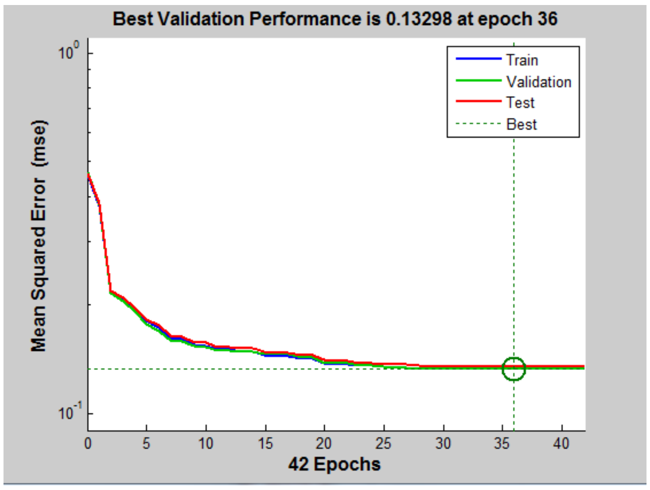


Figure 11. Network performance for Scenario1 (URLs from: List1 & List 2.5 & Malicious URLS)

almost 0.10ms) than Figure 11 (i.e. almost 0.13ms). It means the network reaches to its best point faster when 60% of data have the issue of being benign but having the SQLi attack signature(s). The best validation performance occurs by iteration 44, 43, 36 in Figure 9 to Figure 11 all respectively. The network performance in Scenario1is reasonable because of the following considerations: the final mean square error is small and the test set error and the validation set error have similar characteristics.

The author has also capture the Mean Squared Error (MSE), which is the difference between outputs and targets, and Percentage Error (%E), which is the fraction of the samples which are misclassified, for five sets of experiments in Table5. In Table 5, value 0 means no error for MSE and no misclassifications for %E.

VI. SCENARIO 2: DETECTION, CLASSIFICATION, AND STRESS TEST

In SCENARIO2, which is called DETECTION, CLASSIFICATION, and STRESS TEST, the author focuses on responsibility number two of the proposed NN-based model: identifying the type of SQLi attack for malicious URLs. The SCENARIO2 has also been implemented with controversial data sets. The data sets called controversial as there is a great chance of having false detections i.e. to falsely classify the malicious URLs. To achieve this, the author has considered the same data sets of 25,000 URLs from SCENARIO1, Table 4. For Scenario2, the training and testing/validating elements of the NN model are defined as follows.

1) Scenario2: Training Elements

- "Input Matrix": this matrix is a logical *25,000 x 32* matrix where *25,000* represent the total number of the URLs in Scenario2 and *32* represents the size of the assigned vectors to the SQLi attack.

TABLE5. (MSE) AND (%E) IN SCENARIO 1

Phase	Samples	MSE	%E
URLs from: List1 & List 2.1 & Malicious URLS			
Training	17500	2.85060e-2	4.97142e-0
Validating	3750	2.77257e-2	5.33333e-0
Testing	3750	2.64664e-2	4.85333e-0
URLs from: List1 & List 2.2 & Malicious URLS			
Training	17500	5.37746e-2	10.01714e-0
Validating	3750	5.27890e-2	9.35999e-0
Testing	3750	5.32295e-2	10.58666e-0
URLs from: List1 & List 2.3 & Malicious URLS			
Training	17500	8.35546e-2	14.94857e-0
Validating	3750	8.34291e-2	15.19999e-0
Testing	3750	8.47232e-2	15.09333e-0
URLs from: List1 & List 2.4 & Malicious URLS			
Training	17500	1.04517e-1	19.67999e-0
Validating	3750	1.05754e-1	19.94666e-0
Testing	3750	1.08780e-1	21.60000e-0
URLs from: List1 & List 2.5 & Malicious URLS			
Training	17500	1.33045e-1	24.84571e-0
Validating	3750	1.32979e-1	24.95999e-0
Testing	3750	1.35371e-1	25.78666e-0

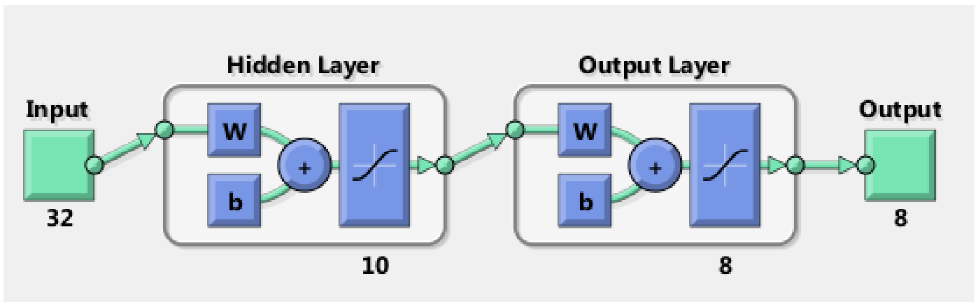


Figure 12. Network Architecture for the Neural Network component of the proposed model (Scenario2)

- "Target Matrix": this matrix is a logical 25,000 x 8 matrix where *25,000* represent the total number of the URLs in Scenario2 and *8* represent the type of the SQLi attack.
- Fitness network: this is the NN model with *10* hidden layers where 70%, 15%, and 15% of the data from 'Input' and 'Target' matrices are employed for training, validating and testing, all respectively.

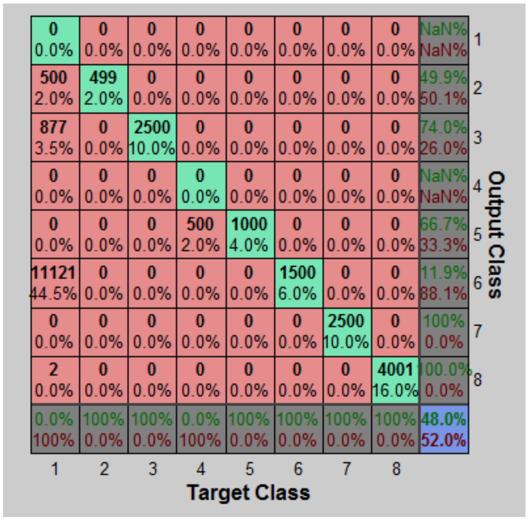
2) Scenario2:Validating/Testing Elements

- "Sample Matrix": this matrix is a logical *n x 32* matrix contains sample *n* data from the "Input Matrix".
- "Output Matrix": this matrix is a logical *n x 8* matrix contains output data for the data represented in "Sample Matrix". The trained NN model predicts the output value, which is the SQLi attack type, for each URL in the "Sample Matrix". These predictions will be stored in the "Output Matrix".

In Scenario2, the NN model has 10 hidden layers, 32 input features, 8 output layer, and 8 output features, Figure 12. For Scenario2, the results have been captured, represented, and analyzed as follows.

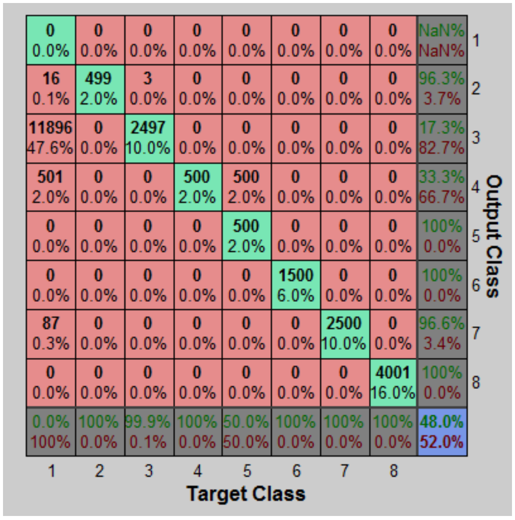
The confusion matrices of Scenario2 for all three phases are captured in Figure 13 to Figure 17. Moving from Figure 13 to Figure 17, as it was expected before, the author notices that the overall percentage of the accuracies is gradually reduced from 48.0% to 42.0% and the overall percentage of inaccuracies is gradually increased from 52.0% to 58.0%, both respectively. The gradual decrement in the overall percentage of accuracies and the gradual decrement in the overall percentage of the inaccuracies are due to the nature of the data in each set of experiments. This means the more difficult set of data, for instance when there is 100% benign URLs with SQLi attack signature(s), will produce the lowest accuracies and the highest inaccuracies in the decision making process, Figure 17.

The network performance for three phases in the five sets of experiments for Scneario2 is captured in Figure 18 to Figure 20. Moving from Figure 18 to Figure 20, as it was expected, the author notices that the network performance in Scenario2 is gradually decreased from 0.006ms in Figure 18 to 0.022ms in Figure 20 by almost 0.002ms. Obviously, this is because of the nature of the data that regularly gets tougher which means the network takes longer to reach its best performance. The best validation performance occurs by iteration 150, 78, 82 for Figure 18 to Figure 20 all respectively. The network performance in Figure 18 to Figure 20 is reasonable for each set of experiment because of the following considerations: the final mean square error is



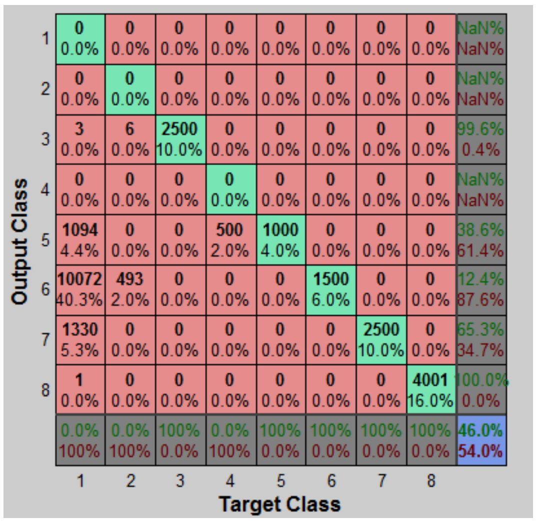
Output Class \ Target Class	1	2	3	4	5	6	7	8	
1	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN% NaN%
2	500 2.0%	499 2.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	49.9% 50.1%
3	877 3.5%	0 0.0%	2500 10.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	74.0% 26.0%
4	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN% NaN%
5	0 0.0%	0 0.0%	0 0.0%	500 2.0%	1000 4.0%	0 0.0%	0 0.0%	0 0.0%	66.7% 33.3%
6	11121 44.5%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1500 6.0%	0 0.0%	0 0.0%	11.9% 88.1%
7	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	2500 10.0%	0 0.0%	100% 0.0%
8	2 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	4001 16.0%	100.0% 0.0%
	0.0% 100%	100% 0.0%	100% 0.0%	0.0% 100%	100% 0.0%	100% 0.0%	100% 0.0%	100% 0.0%	48.0% 52.0%

Figure 13. Confusion matrix for three phases (URLs from: List1 & List 2.1 & Malicious URLS in Scenario2)



Output Class \ Target Class	1	2	3	4	5	6	7	8	
1	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN% NaN%
2	16 0.1%	499 2.0%	3 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	96.3% 3.7%
3	11896 47.6%	0 0.0%	2497 10.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	17.3% 82.7%
4	501 2.0%	0 0.0%	0 0.0%	500 2.0%	500 2.0%	0 0.0%	0 0.0%	0 0.0%	33.3% 66.7%
5	0 0.0%	0 0.0%	0 0.0%	0 0.0%	500 2.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%
6	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1500 6.0%	0 0.0%	0 0.0%	100% 0.0%
7	87 0.3%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	2500 10.0%	0 0.0%	96.6% 3.4%
8	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	4001 16.0%	100% 0.0%
	0.0% 100%	100% 0.0%	99.9% 0.1%	100% 0.0%	50.0% 50.0%	100% 0.0%	100% 0.0%	100% 0.0%	48.0% 52.0%

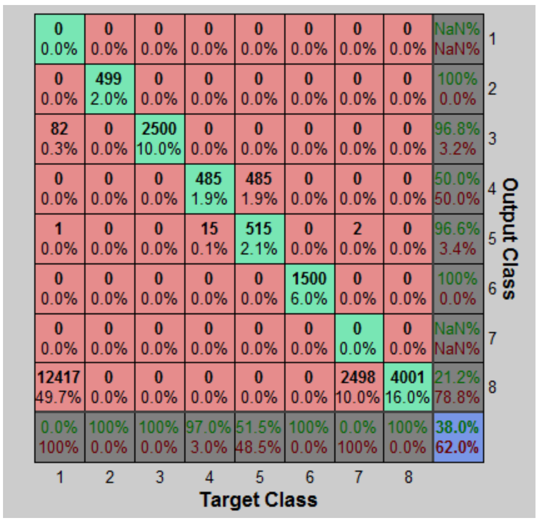
Figure 14. Confusion matrix for three phases (URLs from: List1 & List 2.2 & Malicious URLS in Scenario2)


Output Class

	1	2	3	4	5	6	7	8	
1	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN% NaN%
2	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN% NaN%
3	3 0.0%	6 0.0%	2500 10.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	99.6% 0.4%
4	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN% NaN%
5	1094 4.4%	0 0.0%	0 0.0%	500 2.0%	1000 4.0%	0 0.0%	0 0.0%	0 0.0%	38.6% 61.4%
6	10072 40.3%	493 2.0%	0 0.0%	0 0.0%	0 0.0%	1500 6.0%	0 0.0%	0 0.0%	12.4% 87.6%
7	1330 5.3%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	2500 10.0%	0 0.0%	65.3% 34.7%
8	1 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	4001 16.0%	100.0% 0.0%
	0.0% 100%	0.0% 100%	100% 0.0%	0.0% 100%	100% 0.0%	100% 0.0%	100% 0.0%	100% 0.0%	46.0% 54.0%

Target Class

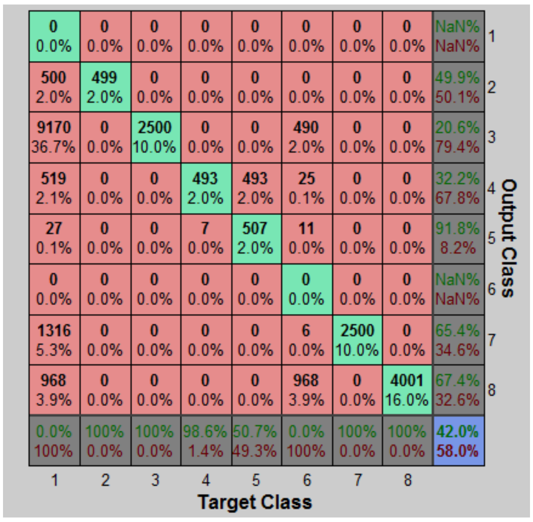
Figure 15. Confusion matrix for three phases (URLs from: List1 & List 2.3 & Malicious URLS in Scenario2)


Output Class

	1	2	3	4	5	6	7	8	
1	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN% NaN%
2	0 0.0%	499 2.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%
3	82 0.3%	0 0.0%	2500 10.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	96.8% 3.2%
4	0 0.0%	0 0.0%	0 0.0%	485 1.9%	485 1.9%	0 0.0%	0 0.0%	0 0.0%	50.0% 50.0%
5	1 0.0%	0 0.0%	0 0.0%	15 0.1%	515 2.1%	0 0.0%	2 0.0%	0 0.0%	96.6% 3.4%
6	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1500 6.0%	0 0.0%	0 0.0%	100% 0.0%
7	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN% NaN%
8	12417 49.7%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	2498 10.0%	4001 16.0%	21.2% 78.8%
	0.0% 100%	100% 0.0%	100% 0.0%	97.0% 3.0%	51.5% 48.5%	100% 0.0%	0.0% 100%	100% 0.0%	38.0% 62.0%

Target Class

Figure 16. Confusion matrix for three phases (URLs from: List1 & List 2.4 & Malicious URLS in Scenario2)


Output Class

	1	2	3	4	5	6	7	8	
1	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN% NaN%
2	500 2.0%	499 2.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	49.9% 50.1%
3	9170 36.7%	0 0.0%	2500 10.0%	0 0.0%	0 0.0%	490 2.0%	0 0.0%	0 0.0%	20.6% 79.4%
4	519 2.1%	0 0.0%	0 0.0%	493 2.0%	493 2.0%	25 0.1%	0 0.0%	0 0.0%	32.2% 67.8%
5	27 0.1%	0 0.0%	0 0.0%	7 0.0%	507 2.0%	11 0.0%	0 0.0%	0 0.0%	91.8% 8.2%
6	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	NaN% NaN%
7	1316 5.3%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	6 0.0%	2500 10.0%	0 0.0%	65.4% 34.6%
8	968 3.9%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	968 3.9%	0 0.0%	4001 16.0%	67.4% 32.6%
	0.0% 100%	100% 0.0%	100% 0.0%	98.6% 1.4%	50.7% 49.3%	0.0% 100%	100% 0.0%	100% 0.0%	42.0% 58.0%

Target Class

Figure 17. Confusion matrix for three phases (URLs from: List1 & List 2.5 & Malicious URLS in Scenario2)

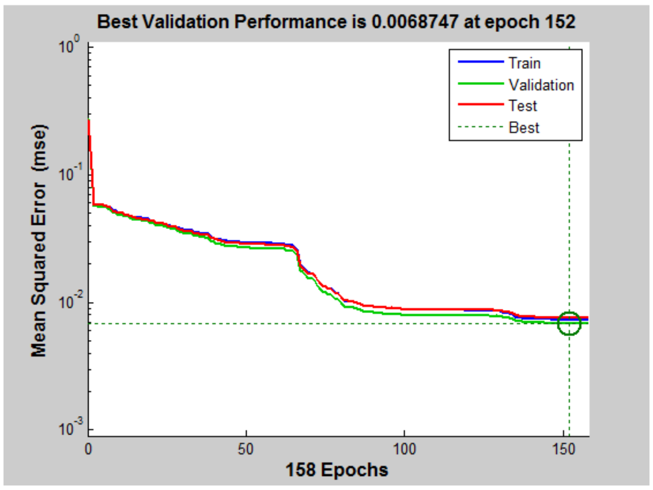


Figure 18. Network performance (URLs from: List1 & List 2.1 & Malicious URLS in Scenario2)

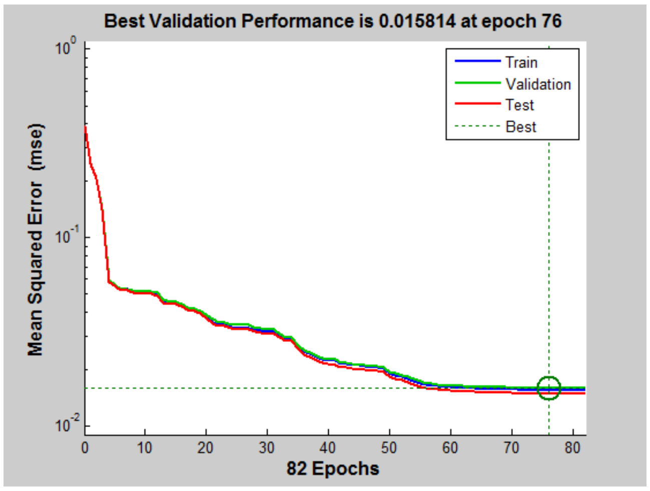


Figure 19. Network performance (URLs from: List1 & List 2.3 & Malicious URLS in Scenario2)

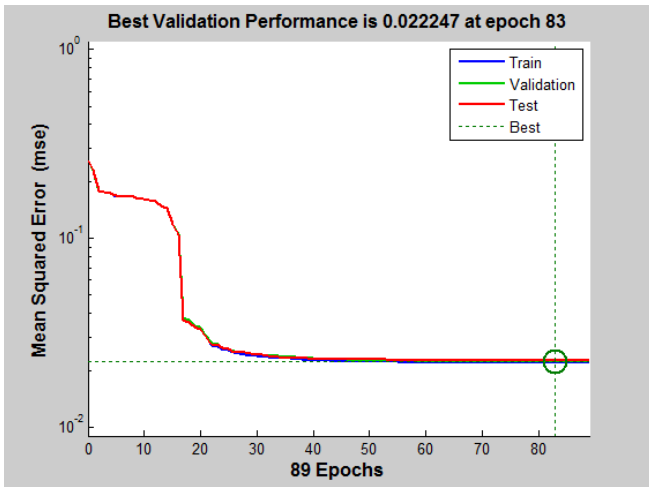


Figure 20. Network performance (URLs from: List1 & List 2.5 & Malicious URLS in Scenario2)

Table6. (MSE) and (%E) in Scenario 2

Phase	Samples	MSE	%E
URLs from: List1 & List 2.1 & Malicious URLS			
Training	17500	7.33524e-3	26.82285e-0
Validating	3750	6.87468e-3	27.58666e-0
Testing	3750	7.57969e-3	27.23999e-0
URLs from: List1 & List 2.2 & Malicious URLS			
Training	17500	8.26756e-3	26.88857e-0
Validating	3750	8.52256e-3	27.40000e-0
Testing	3750	8.83980e-3	27.20000e-0
URLs from: List1 & List 2.3 & Malicious URLS			
Training	17500	1.54859e-2	28.99999e-0
Validating	3750	1.58143e-2	29.08000e-0
Testing	3750	1.48089e-2	28.89333e-0
URLs from: List1 & List 2.4 & Malicious URLS			
Training	17500	2.51881e-2	36.82571e-0
Validating	3750	2.49600e-2	37.65333e-0
Testing	3750	2.46622e-2	37.15999e-0
URLs from: List1 & List 2.5 & Malicious URLS			
Training	17500	2.18843e-2	33.04857e-0
Validating	3750	2.22466e-2	32.30666e-0
Testing	3750	2.25724e-2	33.46666e-0

small and the test set error and the validation set error have similar characteristics.

The MSE and %E for five sets of experiments has been also captured for Scenario2 in Table6.

VII. CONCLUSION

In this paper, we further investigated the performance of our previous proposal from [14-16]. We measured the effectiveness of our previous proposal in terms of accuracy, true-positive rate, and false positive-rate through two scenarios: Scenario1 and Scenario2. In both scenarios, we took into account controversial data sets as there was a great chance for the previous proposal to provide false detections. For instance, it was a great chance to falsely detect the malicious URLs as benign and vice versa in Scenario1 and falsely classify the malicious URLs. For both scenarios, the results were captured, represented, and analyzed in two groups of: confusion matrix and network performance. Addressing the captured results the proposed neural network model for detection and classification of the SQLi attack showed a good performance in terms of accuracy, true-positive rate as well as false-positive rate.

ACKNOWLEDGMENT

The author would wish to acknowledge the support of the Edinburgh Napier University for funding this work.

REFERENCES

[1] Halfond, W. G., Viegas, J., & Orso, A. (2006, March). A classification of SQL-injection attacks and countermeasures. In Proceedings of the IEEE International Symposium on Secure Software Engineering (Vol. 1, pp. 13-15). IEEE.

[2] Rawat, R., & Raghuwanshi, S. (2012). SQL injection attack Detection using SVM. International Journal of Computer Applications, 42(13), 1-4.

[3] Gould, C., Su, Z., & Devanbu, P. (2004, May). JDBC checker: A static analysis tool for SQL/JDBC applications. In Proceedings of the 26th International Conference on Software Engineering (pp. 697-698). IEEE Computer Society.

[4] Lambert, N., & Lin, K. S. (2010, July). Use of Query Tokenization to detect and prevent SQL Injection Attacks. In Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on (Vol. 2, pp. 438-440). IEEE.

[5] Avireddy, S., Perumal, V., Gowraj, N., Kannan, R. S., Thinakaran, P., Ganapthi, S., ... & Prabhu, S. (2012, June). Random4: an application specific randomized encryption algorithm to prevent SQL injection. In Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on (pp. 1327-1333). IEEE.

[6] Shanmughaneethi, S. V., Shyni, S. C. E., & Swamynathan, S. (2009, December). SBSQLID: securing web applications with service based SQL injection detection. In 2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies (pp. 702-704). IEEE.

[7] Zhang, K. X., Lin, C. J., Chen, S. J., Hwang, Y., Huang, H. L., & Hsu, F. H. (2011, November). TransSQL: a translation and validation-based solution for SQL-injection attacks. In Robot, Vision and Signal Processing (RVSP), 2011 First International Conference on (pp. 248-251). IEEE.

[8] Halfond, W. G., & Orso, A. (2005, November). AMNESIA: analysis and monitoring for NEutralizing SQL-injection attacks. In Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering (pp. 174-183). ACM.

[9] McClure, R. A., & Krüger, I. H. (2005, May). SQL DOM: compile time checking of dynamic SQL statements. In Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on (pp. 88-96). IEEE.

[10] Boyd, S. W., & Keromytis, A. D. (2004, June). SQLrand: Preventing SQL injection attacks. In Applied Cryptography and Network Security (pp. 292-302). Springer Berlin Heidelberg.

[11] Huang, Y. W., Huang, S. K., Lin, T. P., & Tsai, C. H. (2003, May). Web application security assessment by fault injection and behavior monitoring. In Proceedings of the 12th international conference on World Wide Web (pp. 148-159). ACM.

[12] Su, Z., & Wassermann, G. (2006, January). The essence of command injection attacks in web applications. In ACM SIGPLAN Notices (Vol. 41, No. 1, pp. 372-382). ACM.

[13] Top 10 2013-Top 10. (2013, October). Retrieved June 17, 2016, from https://www.owasp.org/index.php/Top_10_2013-Top_10

[14] Moradpoor, N. (2014). Employing Neural Networks for the detection of SQL injection attack. In Proceedings of the 7th International Conference on Security of Information and Networks, September 9-11, Glasgow, UK. ACM.

[15] Moradpoor, N. (2015). A Pattern Recognition Neural Network Model for Detection and Classification of SQL Injection Attacks. World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering, 9(6), 1355-1365.

[16] Moradpoor, N. (2015, September). SQL-IDS: evaluation of SQLi attack detection and classification based on machine learning techniques. InProceedings of the 8th International Conference on Security of Information and Networks (pp. 258-266). ACM

[17] SQL introduction. Retrieved June 17, 2016, from http://www.w3schools.com/sql/sql_intro.asp

[18] Internet, A. Top sites for countries. Retrieved June 17, 2016, from http://www.alexa.com/topsites/countries

[19] Google. Retrieved June 17, 2016, from http://www.google.co.uk

[20] Group, T. P. (2016, June 9). PHP 7.1.0 alpha 1 released. Retrieved June 17, 2016, from http://www.php.net

[21] Retrieved June 17, 2016, from http://www.www.en.wikipedia.org/wiki/European_union

[22] MathWorks, T. (1994). MathWorks – makers of MATLAB and Simulink - MathWorks United Kingdom. Retrieved June 17, 2016, from http://www.mathworks.co.uk