International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:4, No:4, 2010

Artificial Neural Network based Web Application Firewall for SQL Injection

Asaad Moosa

***Abstract*—**In recent years with the rapid development of Internet and the Web, more and more web applications have been deployed in many fields and organizations such as finance, military, and government. Together with that, hackers have found more subtle ways to attack web applications. According to international statistics, SQL Injection is one of the most popular vulnerabilities of web applications. The consequences of this type of attacks are quite dangerous, such as sensitive information could be stolen or authentication systems might be by-passed. To mitigate the situation, several techniques have been adopted. In this research, a security solution is proposed using Artificial Neural Network to protect web applications against this type of attacks. The solution has been experimented on sample datasets and has given promising result. The solution has also been developed in a prototypic web application firewall called ANNbWAF.

***Keywords*—**Artificial Neural Networks ANN, SQL Injection, Web Application Firewall WAF, Web Application Scanner WAS.

I. Introduction

TO provide maximum security for web applications, there are specific solutions should be implemented. One of these solutions is Web Application Firewalls (WAF). Most WAFs are based on filtering incoming user requests against a set of predefined rules and signatures. The ability of pattern matching is mainly achieved using regular expressions, such as in ModSecurity; the most famous WAF [19]. However, with the rapid development of web applications, the number of threats and defined attacks signatures is dramatically increasing. Accordingly, traditional pattern matching techniques (particularly regular expressions) are not effective anymore. There is an urgent need to adopt a new pattern matching technique that tackles the requirements of the current stage of security measures.

By listing the security requirements of the new approach, it will be easier to make the choice. Firstly, the new approach should be scalable. This means that it should function well with the increasing number of rules and signatures of known bad incoming requests. Also, it has to be easily updated and user friendly (less complexity than regular expressions). Moreover, it has to be able to deal with the dynamic nature of web application attacks and signatures, including its complicated patterns, such as SQL injection signatures with all possible evasion techniques. More importantly, the time of filtering incoming requests should not affect the performance of the web application.

By investigating possible engineering approaches and metaphors, there is a biologically inspired computing approach precisely matches all the requirements listed above. This approach is the Artificial Neural Networks (ANN).

This paper will go through the concept of artificial neural networks and how to apply it in a form of a web application firewall. To focus on proving the concept of utilizing ANN in a web application security framework, the case study that was investigated in this research is mainly SQL injection attacks, with little involvement of Cross-site Scripting attacks. There are two algorithms have been developed, implemented, examined and carefully evaluated in this research.

Finally, this research is a serious step to adopt ANN approach in the web applications, after the significant success of ANN in network security solutions, such as intrusion detection systems as it will be shown in the literature of this research. The next section will give a brief introduction to the concept of SQL injection and how it can penetrate security solutions using evasion techniques.

II. Introduction to SQL Injection

In the OWASP Top Ten 2007 web application vulnerabilities [25], Injection Flaw was ranked the second most prevalent vulnerability. The priority has jumped to number one most critical vulnerability in OWASP Top Ten 2010 release [26]. This reflects the seriousness of this type of vulnerabilities. In the Injection Flaw family, *SQL Injection* is particularly popular and can cause various consequences in compromising web applications.

Basically, SQL injection attacks occur when web applications directly use user's inputs to build an SQL query to access the backend database without a proper validation on the inputs [36].

To perform SQL injection, hackers can use different techniques. These techniques can be classified into five main categories as will be explained below [9]. *Note: the bold part in each example below is the input entered by the user.*

A. Tautologies

In this type, the attackers inject some SQL token into the user input and cause the selection clause of an SQL query to be true all the time:

Asaad Moosa is with the Centre for Advanced Computing and Emerging Technologies, School of Systems Engineering, Philip Lyle Building, 5th Floor, Whiteknights Campus, University of Reading, Reading, Berkshire, RG6 6BX, United Kingdom. (phone: +447792399468; fax: +441183785224; e-mail: a.moosa@reading.ac.uk).

```
Select * from users where username='admin' or
1=1 --' and password='';
```

B. Union Queries

In this type, the attackers inject a UNION query into the SQL query to get more data:

```
Select bookTitle, ISBN from books where bookID
= 1 UNION Select "hack", balance from accounts
where accNo = 3456 --;
```

C. Piggyback Queries

In this type, the attackers inject additional statements to execute for hacking purpose:

```
Select * from users where username=''; drop
table accounts -- and password=''
```

D. Malformed Queries

This type is based on the error message returned from the web server to find more information about the database:

```
Select * from books where
bookID=convert(int,(select top 1 name from
sysobjects where xtype ='u'));
```

E. Inference

This type of attack often is based on different response-time of the web server to discover other information about the database:

```
Select * from users where username='hello1';
select if( user() like 'root@%',
benchmark(1000000,sha1('test')), 'false' ); --
' and password=''
```

F. Alternate Encoding

This technique is used to by-pass the defending scheme that escapes special characters (such as quotes, dashes, etc.) or some keywords:

```
Select * from books where bookID=1;
exec(char(0x730065006c00650063007400200040004000760065007200730069006f006e00);
```

This runs *sp_msdropretry* [`foo drop table logs select * from sysobjects`], [`bar`].

The various techniques of SQL injection listed above are used by hackers to achieve different purposes: bypassing a login system, modifying a table in a database (using some SQL queries, such as insert, delete, update, etc), shutting down SQL Server, getting database information from the returned error message or inference, or executing stored procedures. Moreover, this can lead to further damages. For instance, after getting the login credentials of the administrator/root of a website through updating the database or abstracting valuable information from the error message, the hacker can login with the administrator privilege and perform sensitive actions. The next section will shed the light on advanced techniques used by hackers to bypass traditional security defense systems.

III. SQL Injection Evasion Techniques

With the evolution of SQL injections, more awareness has been gained by website administrators and developers. The elementary techniques that have been used to prevent web applications against SQL injections include applying some input validations using pattern matching on common SQL keywords such SELECT, EXEC, INSERT, etc. However, hackers can easily overcome these filtering barriers by using some evasion techniques [13], [18] as it will be explained below.

Firstly, SQL evasion can be achieved using a technique called the *C-like comment* (/* some comments*/). For instance, a website uses a Web Application Firewall to reject any HTTP request that has a pattern matches the keyword "UNION" followed by one or more spaces followed by the keyword "SELECT". The hacker can simply inject the string "*/* anything goes here *//*" between UNION and SELECT keywords. The response of different database management systems (DBMS) varies from to another, though, the evasion succeeds. Three DBMS examples will be evaluated for this purpose: Microsoft SQL Server, Oracle, and MySQL.

If the backend database is Microsoft SQL Server or Oracle, then the hacker can bypass the firewall and the injected string will succeed using this evasion approach [13]. The reason is because after the injected string passes through the Firewall, the string "*/* anything goes here *//*" is replaced by a single space by the SQL parser and the injected sequence becomes a legitimate SQL statement. One might argue that the developers can perform the validation in a way that the comment string is replaced by a single space before trying to match the injected string with the pattern above. However, even with doing so, the hacker might overcome this validation and probably has the injected SQL string executed if the backend database server is MySQL. In MySQL, the attacker can do so by submitting a string similar to this "…*UNI/* anything */ON/* anything */ SE/* anything */LE/* anything */CT* …". Assuming that the firewall will replace any C-like comment by a single space, the injected string becomes "…*UNI ON SE LE CT*..." with some spaces between the keywords and the string, which will not be considered as a match against the pattern. Therefore, the injected string will successfully pass through. When the injected string is parsed by MySQL parser, the C-like comments are simply omitted. Then the injected string becomes "…*UNION SELECT*…" which is a genuine SQL statement that might get executed causing a successful SQL injection.

The second approach of evasion, which is so-called *string concatenation*, allows the attacker to achieve SQL injections

by overcoming the firewall with pattern matching like the one above. For instance, in the case of Microsoft SQL Server, the hacker can submit a string like "*…EXEC ("IN" + "SERT" + " IN" + "TO…"…)*". Since the EXEC command of MS SQL Server can be used to execute an SQL statement from a string, which might be a result of a string concatenation, the string "*INSERT INTO…*" probably gets executed [13], [34]. Consequently, even if the developers have a validation mechanism to reject any input with the pattern "*INSERT INTO*", the hacker can still pass through it with the described technique.

Besides the above two techniques, the attacker can also exploit other similar techniques such as the variation of the number of whitespace, using char() function to build a string to bypass the firewall regardless the pattern matching technique used [13], [34].

IV. Related Work

As the number of web application security vulnerabilities and incidents increases day by day, there have been some solutions to mitigate the situation. These solutions are often in a form of a Web Application Scanner (WAS) and a Web Application Firewall (WAF).

A WAS is computer software that search for web applications' vulnerabilities before these web applications are published online [36]. Since a WAS is not meant to work as a real-time filtering mechanism for incoming traffic, it does not affect the performance of the web applications. However, a WAS cannot protect the web applications on the fly and requires modification on the code of the web applications which is often laborious and tedious. Besides, if the source of the web applications is not accessible when the test is running after publishing the web application, then the detected vulnerabilities might not be mitigated.

To protect web applications on the fly, web application firewalls are used. A WAF is a program used to protect a web application in real-time whilst the application is running. The WAF sits between the web-client and the web application, intercepts the flow of information and performs canonical validations based on certain predefined policies. The security policies might exploit positive security model or negative security model or both. Some other security policies might be based on anomaly detection (profile-based) techniques. These three techniques will be discussed briefly.

IV.1. Positive Security Model of WAF

Positive Security model can be described as "Deny All but Known Good". Its philosophy is to define a white-list and if the information to be checked is matched, then it is allowed to pass the WAF and access the web application, otherwise, it is rejected. The white-list for the web application firewall at the application layer can include information such as URL, parameters, parameters' properties, parameters' contents, HTTP methods, etc.

Microsoft® Intelligent Application Gateway (IAG) is a WAF that mainly focuses on this scheme. Microsoft® IAG allows web administrators to define a list of URL pattern along with the HTTP request methods (i.e. POST, GET, etc.) to be accepted for every individual page in the website. Each URL pattern is automatically written using a regular expression and can match a set of allowed URLs. Other URLs that do not match that set will be rejected. Moreover, Microsoft® IAG WAF allows administrators to define a list of parameters together with its types, its value pattern (also defined by regular expressions), their length, whether they are required or not [39].

From a security perspective, positive logic is the preferred model when comparing between the white-listing and blacklisting (negative security model) [6]. It can help to significantly reduce the number of attacks of web applications with SQL injection and other kinds of attacks that take advantage of the lack of input validation techniques.

However, the positive security model has certain drawbacks. First, this approach is not applicable for large-scale web applications where the white-list is too complicated to generate and to validate all possible parameters in every single page in the web application. It is also not practically suitable for frequently changing Web applications since the mechanism of generating the while-list requires manual efforts. Moreover, it is CPU/Memory consuming for busy web applications where hundreds or thousands of HTTP requests are simultaneously are filtered (i.e. Facebook, MySpace, Amazon, etc.). Finally, and most importantly, there is a high possibility that the white-list itself has vulnerabilities, which means that the white-list does not consider any negative based filtering to validate it against vulnerable signatures. The next section will cover the negative filtering model.

IV.2. Negative Security Model of WAF

Negative Security model defines a blacklist of invalid traffic (negative signatures). If the HTTP flaw matches any pattern in the list, it is rejected; otherwise, it is allowed to pass the WAF and access the web application. The patterns used for SQL injection can be as simple as a list of suspicious characters [2] (such as ', --, #), or SQL keywords [12] (such as SELECT, INSERT, etc.), or as complex as a group of ordered SQL keywords and suspicious characters (often in the form of regular expressions) that can be seen in an SQL statement [19].

There is an approach that is based on suspicious characters can catch many forms of SQL injection signatures. However, it can also cause high false positives (i.e. a legitimate request is incorrectly classified as an attack) because the suspicious characters might also appear in legitimate requests. For instance, a quote (') can appear in a name such as O'Brien. Besides, there are cases that some SQL injection patterns do not contain any of these suspicious characters, which consequently means a high possibility of false negatives (i.e. an attack is not caught) might occur too.

On the other hand, another approach that is based on SQL

keywords (each keyword is considered as a pattern) can also catch significant SQL injection signatures. Nonetheless, the limitation of this approach is that some of the defined SQL keywords might also appear in a legitimate request. For instance, an HTTP request with a POST method can have a parameter with the content of the Body part of the HTTP message "*…I will select the following courses…*". In this example, this request will be considered as an attack, which is incorrect (false positive).

A more advanced approach that is based on a group of ordered SQL keywords and suspicious characters is considered to give much higher accuracy in comparison with the previous two approaches. This approach is used by ModSecurity web application firewall, the most deployed WAF according to a research in 2006 [8]. However, the third solution also has some limitations:

- It is quite complex to write a regular expression like the one above.
- With the new evasion SQL injection techniques like the ones described in Section III, some SQL injection patterns can still go through ModSecurity WAF. To prove this drawback with ModSecurity WAF, some tests have been performed against ModSecurity 2.5.7 with Apache Web Server 2.2.10 and a simple PHP forum using MySQL database. The results revealed that when a C-like comment was inserted between the SQL keywords, ModSecurity could not detect it. This is because C-like comments were replaced by a space. Other evasion techniques like string concatenation (Section III) or variations on theme (e.g. instead of using "*…OR 1 = 1* ", " *'…OR 'Simple' LIKE 'Sim%'* " is used) can also penetrate ModSecurity.
- It might be even more complex to update/modify the regular expression set than incorporating new signatures like the ones above.
- The number of regular expressions probably has to dramatically increase when more and more bad patterns are devised by hackers (e.g. according to N-Stalker WAS [23], the database of negative signatures at the time of this writing has around 39,000 signatures). This can lead to increasing the processing time that is utilised by the WAF and will consequently degrade the performance of the web application, which is unacceptable in most cases.

IV.3. Anomaly Detection Model of WAF

In this security model, there is a training phase to be accomplished before the WAF can start functioning to protect web applications against various injection flaws. During the training phase, a profile of normal traffic behavior is built based on either static source code analysis [22], [37] or dynamic checking [33] when the application is safely running in the testing mode. During production phase, the SQL queries submitted to the web application are captured and compared with the profile. If there is any anomaly, then an alarm is generated and the query is rejected based on the predefined policies in the profile.

The advantage of this approach is that it can give high detection rate with low false positive rate. Nonetheless, the disadvantage is it allows the web application to run for a while before the injection flaw can be detected. It requires the modification when the backend database server is changed so that it can capture SQL queries emitted from the web application. And most of all, it is too specific to SQL injections and probably could not be extended to include other types of injection flaws vulnerabilities, due to the lack of input validation.

IV.4. Discussion

WASs can be used to detect vulnerabilities of web applications but cannot function on the fly. WAFs are the common solution to protect web application against various application-layer attacks. WAFs based on positive logic are useful but are not enough and often need to work with negative logic. WAFs based on anomaly detection are often too specific to SQL injection and might not be able to be extended to include other types of injection flaws that take advantage of the lack of input validation. WAFs based on negative security model are the more popular.

Negative logic WAFs can be quite simple when they perform input validation using suspicious characters only or separate SQL keywords only. However, the accuracy of these WAFs might not be as desired. Negative logic WAFs that are based on a group of ordered SQL keywords and suspicious characters using regular expression are believed to provide much more accuracy. Nonetheless the limitations of these WAFs are the complexity in writing the list of regular expressions and keeping it up-to-date especially when the set of known bad patterns dramatically grows. This will lead to a crucial need to investigate a new approach to be adopted to overcome these limitations.

The new approach that is considered in this research is based on the ability of the Artificial Neural Networks concept to perform pattern recognition when it's properly trained. However, first, the concept of Artificial Neural Networks will be explored, which will be covered in the next section.

V. Artificial Neural Networks

An Artificial Neural Network (ANN) is a massively parallel distributed processor consists of a set of neurons interconnected to each other [10]. Like a human brain, an ANN has the ability to learn through a training process to obtain knowledge and makes that knowledge available for later use.

The basic component of an ANN is the neuron. Each neuron has three important components (fig. 1): a set of synaptic connections (which are represented by a set of synaptic weights and bias, i.e. w_{ki} and b_k); a propagation function (Σ) which is a linear combination between the input elements modified by the set of synaptic weights and bias; and an activation function (φ) which takes the output of the propagation function as its input and generates the output of

the neuron. It is the set of synaptic weights and bias that stores the knowledge acquired during the learning phase.

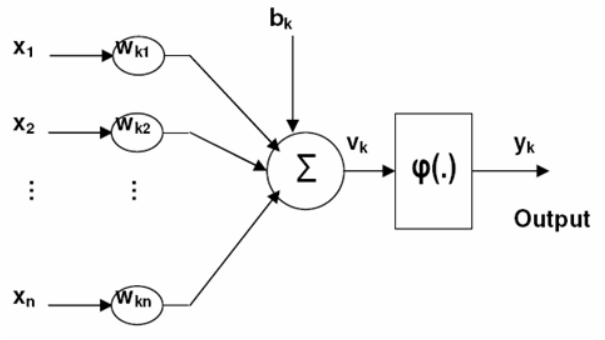

Fig. 1 A Model of a Neuron [10]

The way neurons are connected to one another will define the architecture of an ANN. In this research, Multilayer Feedforward Networks (MLN) was used. The architecture of MLNs is demonstrated in fig. 2.

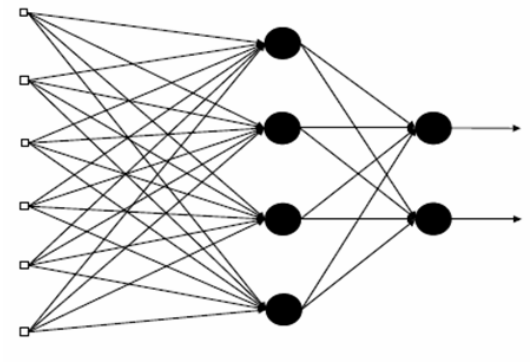

Fig. 2 A Multilayer Feedforward Network (MLN) [10]

With the learning ability, ANN can be trained to perform different engineering tasks. Some of the tasks that can be identified are: pattern recognition, pattern association, function approximation, control systems, filtering, and beam forming [10]. Among these different learning tasks, pattern recognition is the one of interest in this research. Pattern recognition is a process in which a pattern or input is assigned to one of a predefined category or a class.

There are some algorithms that can be used to train an ANN for a pattern recognition task, such as: Back Propagation, Radial-basis Function, and Support Vector Learning, etc. Among them, *Back Propagation* is the algorithm that is specifically devised to train a multilayer perceptron. The algorithm has been implemented in Matlab® [15], which is a popular tool to train ANNs.

MLNs trained with Back Propagation have been used in different fields such as Intrusion Detection Systems [20], [21], [27] and Image Processing [1], [11]. The classification accuracies in all these applications are higher than 90%, especially the application of MLN in an intrusion detection approach has an accuracy of 99.25% [21].

The success of ANN in intrusion detection systems has motivates this research to investigate a new solution for the challenging limitations of Web Application Firewalls (WAF). More importantly, with ANNs the answer for a scalable solution for WAFs can be found based on some of the instinct features of the ANNs [10]: The ability to learn and store the empirical knowledge; the nonlinearity of the ANN; the ability to generalize the solutions; the ability to adapt when the context changes; the computational performance; and the massively parallel structure of the ANN.

VI. Proposed Solution

The proposed solution gains the motivation from applying ANN into intrusion detection systems successfully. The proposed Artificial Neural Network-based WAF (ANNbWAF) solution aims to produce a high accuracy in detecting SQL injection attacks without much loss in the performance when the number of negative patterns increases as the case of pattern matching based on regular expressions (e.g. ModSecurity [19]). The solution also has the potential to be extended to include other types of attacks that take advantage of the lack of input validation such as Cross Site Scripting (XSS) and other types of Injection Flaws. In this research, some experiments for XSS have also been carried out.

For shared web hosting, the ANNbWAF is used for each website separately. In this case, the Web Server with ANN-based WAF, which hosts many websites, integrates different instances of the ANNbWAF that need to be trained and used for each website individually. This arrangement is crucial since each website has its own list of valid HTTP traffic, whereas other websites on the same web server may not allow the same traffic, leading to a policy conflict in filtering web traffic. However, the experiments in this research were tested on a single website hosted on a web server.

The ANN-based WAF does not consider the case of Alternate Encoding methods. Often different Encoding methods (e.g. Base 64, Unicode, UTF-8, Hex, Decimal, etc.) are used to support different web languages and character sets [13], [34]. However, the proposed solution assumes that the web server takes the responsibility of decoding the HTTP requests and the ANNbWAF might only need to perform simple decoding with UTF-8 scheme (it is also the recommended encoding method from the WWW Consortium). The proposed solution can be summarized in fig. 3. The system has two phases: Training phase and Working phase. During the training phase, a set of normal and malicious data is used to train the ANN with Matlab®. The trained ANN is then integrated into a WAF to protect the web application during the working phase.

The experiments have been implemented with two ANN filtering approaches: character-based and keyword-based. The following will explain the two methods, showing the considered data sets, the algorithm of each method and discussing and evaluating the results.

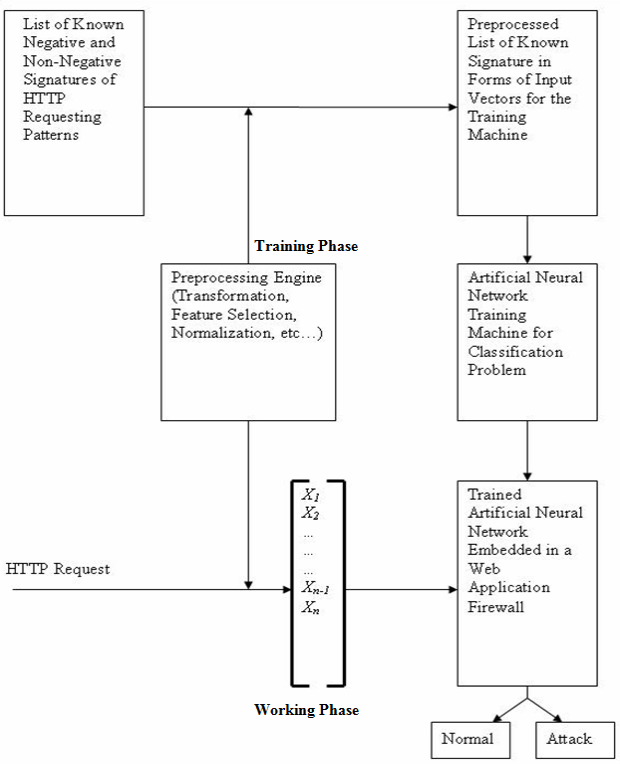

Fig. 3 General Proposed System Design

VI.1. Character-based Method

The approach motivation is based on the observation that some characters and character combinations often have a higher distribution frequency in a malicious content than in a normal one. Besides applying the approach for SQL injection problems, we also examined the approach with XSS attacks. However, since the feature sets (section VI.1.1) are different and with the purpose of reducing the complexity during the preliminary experiments with ANN, an individual ANN was chosen separately for each type of the attacks (SQL injection and XSS). Then, an ANN with both attacks data sets were tested and evaluated.

VI.1.1 Data Set

Since there is no available normalised database of SQL injection signatures along and normal signatures, a set of synthetic data was used as the data sets for the experiment. There have been around 300 SQL injection signatures and around 200 XSS signatures collected from different websites and white papers [3], [4], [5], [7], [13], [14], [16], [17], [30], [28], [29], [32].

The normal signatures are either the usernames and email addresses of a group of students of the University of Reading (where this research was made), a set of passwords from the template passwords of John the Ripper password cracker software [24], from some of random search patterns (software names, book titles, song names, technical topics, etc.), and the messages posted on Sun Java forums [30]. The size of a normal signature is 200 characters.

To achieve input validation in this research, parameters from both the query strings and the body of the request are considered (two most popular places for user input). However, this does not mean that the solution cannot be extended to include parameters from other places in the client request, such as cookies, header, etc.

VI.1.2 Data Preprocessing Algorithm

Based on the analysis of the two negative signature sets of SQL injection attack and XSS attacks, some characters and some combinations of both only appear in either one set or another. For this reason, for each Neural Network, the preprocessing step uses a separate character and character combination set as shown in Table-I.

TABLE I
Character and Character Combination Sets

Categories	XSS	SQL
Alphanumeric	[a-z][A-Z][0-9]	[a-z][A-Z][0-9]
Punctuation	<, >, %, &, ;, #, +, =, (,), ., ', ", /, :, {, }, \, -, @, [,], ?, `	<, >, ', ", *, ;, _, (,), =, {, }, @, ., ,, &, [,], +, -, ?, %, #, (,), !, :, \, /
Special Combination	--, &#, //, /*, */, <!, <?, ?>	xp_, @@, sp_, --, 0x, or, /*, */, ||, >>, \\, &#x, &#, ..
White Space	Space, Tab, Line Feed Carriage Return	Space, Tab, Line Feed Carriage Return
Remaining	Every other characters	Every other characters

In this approach, each signature (either normal or attack) is converted to a vector of numbers and a corresponding flag for the class of the vector ('normal' or 'attack'). Each dimension of the vector represents a character or some special character combinations. The value of each dimension is the number of times the corresponding character appears in the signature.

VI.1.3 Experiment

In this research, the available data sets are used to train the ANN with Back Propagation implemented in Matlab®. The training method is Adaptive Learning with Momentum. During the training process, two hidden layers are used and the number of neurons per hidden layers varies from 1 to 3 when the whole data set is used, a nd 1 to 10 when the data set is split into training (450 samples for SQL injection, 360 for XSS) and validation (205 for SQL injection, 158 for XSS) data. For each configuration, the experiment is carried out 10 times; each time it runs with 1000 epochs. When the whole data set in each case is used for training, the highest accuracy that can be obtained is 100%. When the data sets are split, the best results are in Table II:

TABLE II
Best Result on Split Data

Data Set	Best Result
XSS attacks	100% on the training data, 100% on the validation data.
SQL Injection attacks	100%, on the training data, 98.537% on the validation data.

VI.1.4 Evaluation

From the experiment results, it can be noticed that the approach based on character frequency distribution gives quite promising results. However, the very high accuracy is probably because the data sets are relatively small.

To investigate how effective the approach is, a test data with 30 patterns is used with the best configuration on the split SQL data set. The result of this test reveals that when the frequency of the special characters such as quote (‘) or semicolon (;) in the user input is high, the trained ANN often classifies the input as attack, although the input is a normal one, the overall accuracy is 66.67%. This indicates a high level of expected false positives in this approach. This approach could be enhanced and the result could have been better if more data sets are used for training the ANN. However, the test result also suggests that SQL keywords (or HTML tags and script function in the case of XSS) should be taken into account when the Neural Networks are trained to effectively reduce false positives occurrence. Because of this, a new approach is considered in the next section.

VI.2. Keyword-based Method

In the second approach, only SQL injection is considered first. The motivation for the second approach is that in an SQL injection signature, the frequency of the SQL keywords is higher and some keywords often appear together such as INSERT and INTO or UNION and SELECT. Besides, some special characters and character combinations (such as: #, --, /*, etc.) also appear more often together with the SQL keywords in SQL injection signatures.

VI.2.1 Data Set

To achieve the second approach, there has been more analysis carried out on the collected SQL injection signatures. The SQL injection signatures used in the previous approach have been modified and more signatures have been added in. The number of SQL injection signatures for the second approach is 358. Together with 200 non-negative signatures, the data set used for training in this approach has in total 558 signatures.

VI.2.2 Data Preprocessing Algorithm

There are 44 features that are used as the features of the output vector of the preprocessing stage. Each feature is corresponding to an SQL keyword (keyword group), a character (character group), or a special character combination that appears in the content of the signature.

Basically, the preprocessing algorithm can be described as follows: If a keyword is discovered in the signature, its corresponding feature (i.e. keyword or keyword group) will increase the value by 1. So if a keyword appears more than once, its corresponding feature’s value will increase by the number of occurrence. After all keywords have been considered, the character and the character combination are searched in the signature. A character or a character combination appearance will increase the value of the corresponding feature by 1. A space (whitespace, tab, line feed, and carriage return) character will increase the feature “space” value by 1. After that, the other remaining characters will increase the feature “other” by 1 for any appearance of any of them in the signature. The preprocessing algorithm also has to take in to account the C-like comment evasion technique described in Section III during the process of looking for the keywords in signature.

VI.2.3 Experiment

The experiments in the second approach are carried out in a similar way as in the first approach. This time the number of neurons per hidden layer is varied from 1 to 5 and each configuration is experimented 20 times.

When the whole data set is used for training, the best accuracy is 100%. When the data is split into training data (502 samples) and validation data (184 samples), the best accuracy is 100% on both training and validation data.

VI.2.4 Evaluation

In this approach, the experiments also give promising results on the training data set with some configuration gives 100% accuracy on both training data and validation data.

When the same test data that was applied to the first approach for SQL injection (containing some special characters such as ‘, ;), applied to this approach, the result was 100% on both training and testing data. However, the result reveals that some of the samples are correctly classified but some of them are still misclassified. When a signature is chosen to be classified “normal” as it does not contain any SQL keywords or dangerous characters (such as single line comment: #, --) during the preprocessing step before applying it into the ANN, the accuracy can be significantly increased. When this was done, the accuracy on the test data increased from 76.67% to 83.67%.

Besides, it was also found that when the normal data contains SQL keywords such as SELECT, UNION, INSERT, DETELE, UPDATE is applied to the trained ANN, some of the testing data is misclassified as SQL injection.

This is probably because in the training data set, the normal signature is not large enough to cover the input which is normal, but has one or more SQL keywords. In this case, the training data set can be updated followed by retraining the ANN to reduce the rate of false positives. When the ANN is re-trained to incorporate the new data, it could classify the new instances correctly.

The limitation of this approach is that it is based on the appearance of certain SQL keywords together with suspicious characters, but it does not keep the relative order between them. For this reason, if a normal signature contains many keywords and suspicious characters that often appear together in an SQL injection, then it is highly likely to be misclassified although the order of the keywords is not as in a syntax-correct SQL statement. However, it could be considered in the same time a successful approach to avoid evasion traps.

This approach can also be applied to XSS attacks with the keywords to be HTML tags or the JavaScript/VBScript

functions and might have a better result compared to the first approach. Moreover, this approach can be applied to other types of attacks that are currently dealt with by using regular expressions to match one or more keywords in some order. For instance, this approach can be applied to Operating System (OS) Command Injection problems in which the keywords are the OS commands (such as ls, cp, mv, wget, etc), and Local File Inclusion attacks, in which the common signature to be chased is: (../) that is used to access local resources in the server.

VI.3. Summary

The solution for WAF based on ANN in this research has given some promising result to solve serious web application attacks such as injection flaws and particularly SQL injection attacks and XSS attacks. The accuracy of the ANNbWAF can be improved when more signatures are available and a more effective preprocessing algorithm is devised.

VII. Prototype Implementation

Based on the result of the experiments in the previous section, a prototype of the Neural Network-based WAF (ANNbWAF) was designed and implemented. To implement the proposed security framework, Perl Web Server [38] was used instead of Apache for simplicity and as a proof of concept. However, the proposed framework is meant to be working on all web servers including Apache, Microsoft IIS, etc.

Fig. 4 demonstrates a general processing flowchart of the ANNbWAF for an HTTP request to access a page that exists in a directory of the web server and is open for public access on port 80.

First, the request information from the web server is analyzed by the ANNbWAF. The input parameters of the request are then extracted either from the query string in case of an HTTP request with GET method, or from the body of the request in case of an HTTP request with POST method. For each parameter entered by the user, there are two phases of processing:

- In the first phase, the ANNbWAF first needs to examine if the parameter is required to be checked for SQL injection attacks (first phase). If it is required, then the ANNbWAF preprocesses the content of the input parameter using the Input Preprocessing library. The result of the preprocessing step is a feature vector that can be fed into the trained ANN to check for SQL injection attacks. If the classification result from the ANN indicates that the content of the parameter is an SQL injection attack, then the ANN for SQL injection reports the result back to Web Server and stops processing for the request. On the other hand, if the classification result indicates that the content of the parameter is legitimate or the configuration indicates the parameter does not require a check for SQL injection the second phase is performed.
- In the second phase, a similar process is carried out but this time for XSS attacks, in which both the preprocessing algorithm and the ANN for XSS are used. If the classification output of the ANN finds the content of the parameter is an XSS attack, then the result is passed back to the Perl Web Server. Otherwise, if the result indicates the content of the

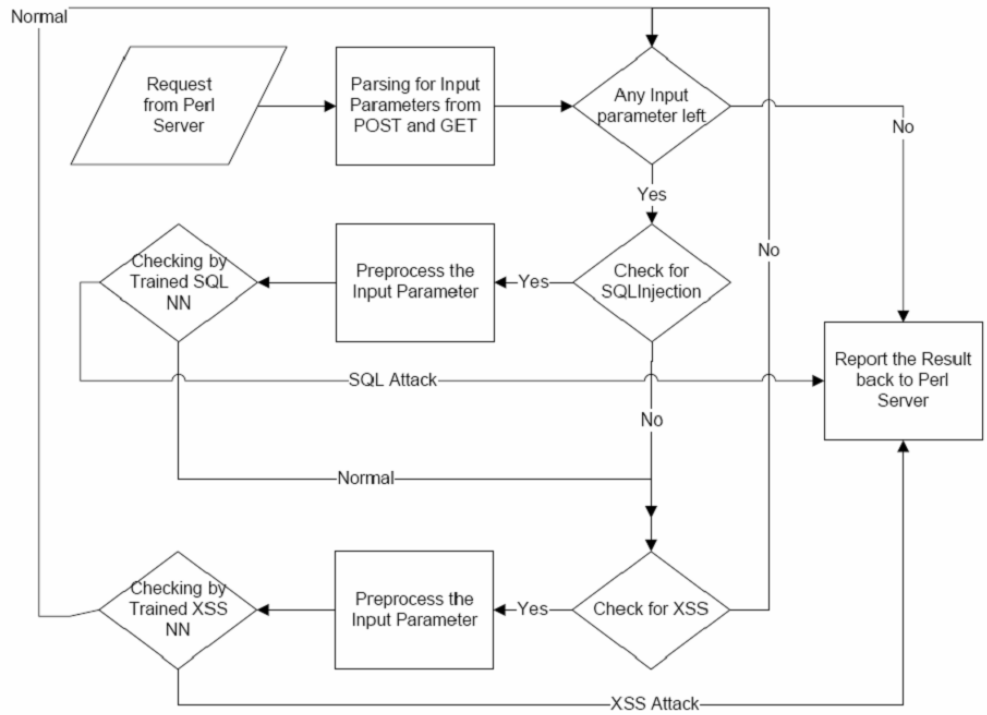

Fig. 4 General Processing Flowchart of ANNbWAF

parameter is legitimate or this parameter does not need to be check for XSS attack, the process continues for any remaining input parameters. If there is no parameter left and no attack is discovered, the ANNbWAF reports this request is a normal one and will be safely passed to the web server.

The web server for the implementation is a simple Perl Web Server [38]. The ANNbWAF is implemented in java programming language. The input preprocessing function is implemented as a separate library in java so that the ANNbWAF can be updated without having to modify the code when the preprocessing algorithm is changed. The configuration of the ANN is not hard-coded directly into the code. Instead, it is stored in a configuration file and the ANNbWAF will read that information during the starting up phase. Moreover, the implementation of the ANNbWAF also incorporates some features from positive security scheme such as defining the list of pages (URL) that should be accepted by the ANNbWAF or the list of parameters of each webpage and the desired way to process each parameter accordingly.

The experiments on the prototype revealed that the accuracy of the ANNbWAF is the same as that of the ANNs which were trained in Matlab® in the training phase. When the configuration of the ANN is changed and even when the input preprocessing library is updated with new algorithm, the ANNbWAF still works correctly after restarting it.

Lastly, the processing time of the ANNbWAF was measured on the second preprocessing algorithm for SQL injection. The result is that the preprocessing time on all 558 signatures (both attacks and normal) is around 250 milliseconds whereas the classification time by the ANN on the same 558 signatures is only less than 1 millisecond. On average, each signature takes ~0.448 millisecond for preprocessing and ~0.002 millisecond for ANN classification. This is a quite promising result in comparison to the performance of many alternative solutions with different algorithms and far better than solutions with regular expression based pattern matching.

VIII. Conclusion

The solution based on an ANN for distinguishing between normal and malicious (SQL Injection) content of the training data gives promising result of both the accuracy and the processing time, especially the second approach which is a keyword-based approach. It has the potential to give a higher accuracy when comparing to the solution that uses some suspicious characters for validation [2] or some SQL keywords separately [12] for the problem of SQL injection. It can also detect bad patterns when some evasion techniques are used (this depends on how the features are extracted from a pattern). Moreover, based on its generalization ability, it might catch new bad patterns based on the SQL keywords and the special characters distribution with the observation that an SQL injection pattern often contain some SQL keywords and/or some special characters (e.g. #, --, /*, etc.). For new patterns that the ANN cannot recognize, the ANN can be re-trained so that it can detect the new patterns without much increase in processing time. The key point here is that the ANN can be re-trained overtime to incorporate more "knowledge" into the ANN. The idea of the solution can be applied for other types of attacks that are currently often dealt with by using regular expressions to match a sequence of keywords and special characters, such as XSS or other types of code injections, etc. Last but not least, the solution can be used in combination with positive logic based filtering as in the prototype implementation.

The solution also has some limitations. The quality of a trained ANN often depends on its architecture and the way the ANN is trained. More importantly, the quality of the trained ANN also depends on the quality of the training data used and the features that are extracted from the data. As in the case in the second approach, with the relatively limited sets of the training data, the resulting ANN seems to be sensitive to a content that has an SQL keyword. However, understanding the nature of the Artificial Neural Networks used in this framework, this approach gives a clear idea of how to solve the current limitations in the most famous web application firewall; ModSecurity [19]. This approach works even better and more precisely when the datasets are relatively tremendous, which indicates an optimal solution to scalability feature of adding new rules and signatures to traditional WAFs that are based on regular expressions. This matches exactly the nature of the Internet security problem, which is the increasing number of web attacks.

IX. Future Work

The proposed prototype is a proof of the concept of how Artificial Neural Networks can replace the traditional approach of web application firewalls filtering mechanisms. It is quite important to realise the nature of the threats in Internet and web applications, and accordingly, a solution that matches the new requirements should take place. This approach is built of the humble success of using ANN in different layers of security, particularly intrusion detection systems. However, there has been no serious attempt to adopt ANN in the Application Layer, which is more vulnerable than other layers. This research is a serious attempt in this direction.

The next step of this research is to generalise this approach on all web application vulnerabilities as mentioned in the OWASP top 10 [25], [26]. This will prepare to a stage of establishing a new open source project of designing a web application firewall completely based on Artificial Neural Networks. More details can be explained here regarding this suggested open source ANNbWAF, but will be left for future research papers.

Regarding the suggested framework, the solution can be extended to include user inputs from any possible HTTP request (not just in the request line the request body), such as headers to have more control over session handling. Also, more protocols can be considered other than HTTP to generalise the solution, such as accepting input from Web Services protocols, like SOAP.

In this research, each signature (normal or malicious) is just the content from one user input. The capacity of filtering can be extended by accepting more inputs that represent the

content of all simultaneous user inputs (that need to be validated) in an HTTP request to reduce the processing time. This is applicable and effectively easy in ANN. By enabling this feature in the framework, it will make this solution extremely useful for large scale web applications, such as Amazon, eBay, Facebook, etc.

The same idea that is applied for SQL injection can also be applied to other types of attacks such as XSS and other types of Code Injection. In the prototype implementation, there is an ANN for XSS using the algorithm in the first approach and this ANN is separated from the ANN for SQL injection. This means there are two separate validations on the two ANNs. This increases the processing time for each request. This situation can be mitigated by using multithreading. The ultimate future solution can be done by training an ANN to be used for all types of web applications attacks and there will be just one validation.

Finally, the training engine can be integrated into the ANNbWAF itself, and by this way there is no need to use Matlab® as an external training engine. This can be achieved by adding a simple graphical user interface to allow the administrators to have a complete control over the ANNbWAF firewall.

ACKNOWLEDGMENT

The author thanks the Centre for Advanced Computing and Emerging Technologies (ACET) in the School of Systems Engineering in the University of Reading, for their help and support.

REFERENCES

[1] F. Ahmadi, Z. M. J. Valadan, H. Ebadi, and M. Mokhtarzade. "The Application Of Neural Networks, Image Processing And CAD-Based Environments Facilities In Automatic Road Extraction And Vectorization From High Resolution Satellite Images". *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. Beijing, pp. 37, 2008.

[2] A. Alfantookh, "An automated universal server level solution for SQL injection security flaw". *International Conference on Electrical, Electronic and Computer Engineering.* pp. 131–135, 2004.

[3] C. Anley, "Advanced SQL Injection In SQL Server Applications". White Paper. Next Generation Security Software, 2002.

[4] C. Anley, "(more) Advanced SQL Injection". White Paper. Next Generation Security Software, 2002.

[5] C. Anley, "Hackproofing MySQL". White Paper. Next Generation Security Software, 2004.

[6] M. Becher, *Web Application Firewalls, Applied Web applications security*. Berlin, 2007.

[7] D. Endler, "The Evolution of Cross-Site Scripting Attacks". White Paper iDEFENSE Incorporation, 2002.

[8] M. Gavin, J.A. Mulligan, L. Koetzle, and S. Bernhardt, *ModSecurity's Web Application Firewall Leads In Deployment Numbers But Lags In Usability.* 2006, [Online] Available: http://www.forrester.com/Research/Document/Excerpt/0,7211,39714,00.html

[9] W. G. J. Halfond, A. Orso, and P. Manolios, "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation". *Software Engineering, IEEE Transactions*. vol. 34, no. 1, pp. 65–81, 2008.

[10] S. Haykin, "Neural Networks, A Comprehensive Foundation". 2nd Edition. New Jersey, USA. Prentice-Hall Inc, 1999.

[11] T. Kubo, M. Obuchi, G. Ohashi, and Y. Shimodaira, "Image processing system for direction detection of an object using neural network". *The 1998 IEEE Asia-Pacific Conference on Circuits and Systems*. pp. 571-574.

[12] Y. Loh, W. Yau, C. Wong, and W. Ho, "Design and Implementation of an XML Firewall". *International Conference on Computational Intelligence and Security*. pp. 1147–1150, 2006.

[13] O. Maor and A. Shulman, "SQL Injection Signature Evasions". White Paper. IMPERA Application Defense Center, 2004.

[14] O. Maor and A. Shulman, "Blind SQL Injection". *Imperva*. [Online] http://www.imperva.com/resources/adc/blind_sql_server_injection.html

[15] Mathworks, *Matlab® The MathWorks™,* [Online] Available: http://www.mathworks.com/

[16] F. Mavituna, "Fast Way to Extract Data From Error Based SQL Injection". *Mavituna* [Online] Available: http://ferruh.mavituna.com/fast-way-to-extract-data-from-error-based-sql-injections-oku/

[17] F. Mavituna, "Fast Way to Extract Data From Error Based SQL Injection". *Mavituna* [Online] Available: http://ferruh.mavituna.com/fast-way-to-extract-data-from-error-based-sql-injections-oku/

[18] F. Mavituna, "SQL Injection Cheat Sheet". *Mavituna* [Online] Available: http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/

[19] Breach Security, *ModSecurity Open Source Web Application Firewall.* [Online] Available: http://www.modsecurity.org/

[20] M. Moradi and M. Zulkernine, "A Neural Network Based System for Intrusion Detection and Classification of Attacks". *Proceeding of the 2004 IEEE International Conference on Advances in Intelligent Systems - Theory and Applications*. Luxembourg. pp.148–153.

[21] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines". *Proceedings of the 2002 International Joint Conference on Neural Networks*. pp. 1702–1707.

[22] M. Muthuprasanna, K. Wei, and S. Kothari, "Eliminating SQL Injection Attacks - A Transparent Defense Mechanism". *Eighth IEEE International Symposium on Web Site Evolution*. pp. 22–32, 2006.

[23] N-Stalker® *N-Stalker Web Application Security Scanner*. [Online] Available: http://www.nstalker.com

[24] Openwall Project, *John the Ripper Password Cracker*. [Online] Available: http://www.openwall.com/john

[25] OWASP, *Top Ten 2007*. [Online] Available: http://www.owasp.org/index.php/Top_10_2007

[26] OWASP, *Top Ten 2010*. [Online] Available: http://www.owasp.org/images/0/0f/OWASP_T10_-_2010_rc1.pdf

[27] J. Ryan, M. J. Lin, and R. Miikkulainen "Intrusion Detection with Neural Networks". *Advances in Neural Information Processing Systems 10*. Cambridge, MA: MIT Press, 1998.

[28] Securiteam, *SQL Injection Walkthrough*. [Online] Available: http://www.securiteam.com/securityreviews/5DP0N1P76E.html.

[29] C. Snake, *XSS (Cross Site Scripting) Cheat Sheet*. [Online] Available: http://ha.ckers.org/xss.html

[30] SunForums, *Sun Forums*. [Online] Available: http://forums.sun.com/index.jspa

[31] Technicalinfo, *HTML Code Injection and Cross-site scripting*. [Online] Available: http://www.technicalinfo.net/papers/CSS.html

[32] Unixwiz, *SQL Injection Attacks by Example*. [Online] Available: http://www.unixwiz.net/techtips/sql-injection.html

[33] M. Valeur, D. Mutz, and G. Vigna, "A Learning-Based Approach to the Detection of SQL Attacks". *Conference on Detection of Intrusions and Malware & Vulnerability Assessment.* 2005.

[34] M. C. Vittie, "SQL Injection Evasion Detection". White Paper. F5 Networks Inc. 2007.

[35] WASC, *Web Hacking Incidents Database*. [Online] Available: http://www.webappsec.org/projects/whid/

[36] WASC, *Web Security Glossary*. [Online] Available: http://www.webappsec.org/projects/glossary/

[37] K. Wei, M. Muthuprasanna, and S. Kothari, "Preventing SQL Injection Attacks in Stored Procedures". *Australian Software Engineer Conference*, Australia, 2006.

[38] The Perl Web Server Project. *Type-O-Serve* [Online] Available: http://perlwebserver.sourceforge.net/

[39] Microsoft Corporation, *Intelligent Application Gateway*. United States: Whale Communications, 2007.