SQL Anomaly Detection and Reconstruction of Queries to Eliminate the Denial of Service

Muhammad Sohaib Yousaf[(1)], M. Sheraz Arshad Malik[(2),] Muhammad Asif[(3*)], Farhat Naz[(1)], Ijaz Ali Shoukat[(4)]

[(1)]Department of Computer Science, Virtual University, Pakistan
[(2)]Department of Information Technology, Government College University Faisalabad, Pakistan
[(3)]Department of Computer Science, National Textile University, Pakistan.
[(4)]Riphah College of Computing, Riphah International University, Faisalabad Campus, Pakistan
*Corresponding Author

Abstract
Different access control mechanisms are used to secure the database against an unauthorized access by either an intruder or extruder. Currently employed mechanisms are either orthodox or they failed considerably to secure the database from authorized and malicious users. There were various techniques, employed previously to detect SQL anomalies and to impede the query from execution. Although the solutions were appropriate to some extent, there was no authentic and stubborn shield available to embark a solid resolution against the reconstruction of queries to minimize the occurrence of denial of service DoS. Various reasons are there for the occurrence of denial of service DoS in the database. The DoS may happen when an SQL query is detected as anomalous and intercepted by the database. Things are even embroiled when some process issues a query and waiting for a reply from the database; certainly, that process has to wait forever for the response by the database. This situation cannot be justified for the organizations which are working with databases in the real time environments. To overcome the issues of DoS, this work proposes a technique of SQL injection detection and purifies those queries from the malicious codes, usually injected by intruders. This work is formed as the next section represents the introduction, literature review, methodology and finally the references section.

Key words:
SQL; Anomaly; DoS; Query

1. Introduction

Secrecy and privacy of data are important to the organizations especially those who have challenging business competitors. The data is vulnerable to several types of attacks that may initialize by the attackers living outside or may come from within the organization. The attempt of data theft can be resulted either from an ex-filtration by insiders or intrusion from the outside world. Insiders are most perilous because they have the valid credentials and hence considered as legitimate users so, they can easily penetrate into the system [1]. No network firewall impedes their malicious activities because the insiders have full permissions of doing any kind of activity. The access control lists provide some constraints, but to a limited extent only because, it does not provide a thorough mechanism for securing the data from insiders that can damage even more scarily [2]. Normally the intrusion activity is done by injecting some malicious SQL code to the actual query, driven by the application program to traverse, insert, update or delete the data. There are various mechanisms to detect the SQL injection attacks. The problem in almost all the approaches was their incapability to clean sweep some hidden and unexploited paths, which creates enough room for the intruders to come in and steal the data. All the novel methods certainly have some advantages when compared with other ones, but the dilemma is their inability to cope with changing environments, i.e. the dynamism which is now considered as a landmark in the computer industry. This dynamism has its clear implications in the form of not only the occurrence of fragility in the sophisticated systems but also in fortifying such systems. This approach uses the signature-based anomaly detection mechanism that matches the patterns of all the SQL queries and application paths that have a possibility of running in this system during execution. A database of these patterns is maintained that are used to compare the SQL queries which are received at the time of query comparison [3]. As a result, the queries that are matched to the query patterns, maintained in a database are considered as benign queries while the rests that do not match are considered as the malicious and assumed to be issued by an intruder. This work not only detects the SQL injections in the queries but also eliminates the augmented parts from that query which is somehow to be executed by the database. The elimination of this malicious code is called as query reconstruction. For reconstruction, a novel mechanism is introduced to find the victim query usually maligned by someone. This victim query is extracted by making a comparison of already formed queries in Backus normal form. The next step is certainly to sanitize the victim query by removing the injected parts from the query. In this regard, a subtraction is made to detract from the inoculated parts. As far as this research is concerned, the following section is composed of the related work, the anomaly detection schemas, then the methodology section, after both these sections the experiment results and experiences are discussed that ensures the successfulness of

Manuscript received May 5, 2018
Manuscript revised May 20, 2018

this study. In the later stages the analysis of the work is written down and at the last, a brief conclusion and future dimensions are also proposed that can be adapted to augment this work for betterment.

2. Literature Review

Anomaly detection is important to the security of databases even if a proper firewall and spam filters are installed in the system. It is just like a house where locks are appropriately installed to all the doors and windows so that no intruder is permitted to enter the house without permission. So in such situations, one may think that there is no need to install the alarms because the house is already protected and secure, but what about the situations where the administrator of the house may forget to punch the lock at the backyard gate. These loopholes are more dangerous than a vigilant security of an open yard. There is a common concern of Buffer Overflows, Intrusions, Denial of Service, Sniffer Attacks and sometimes Application layer attacks. Spam filters and Firewalls are also in place but intrusions are too complex for these firewalls to detect and refrain them from attacking the database. In literature review phase various methods are discussed by which intrusions can be handled by employing different techniques and tools so that intrusion can be detected at the fledgling phase of the attack. In the same way different techniques regarding the reconstruction of the queries are also discussed. The material on the reconstruction is not adequate and available, but a humble attempt is made while writing the related literature of the problem domain.

Dalai and Jena (2017) proposed a novel method for detecting the SQL injection attacks against online applications. The methodology behind this work was the extraction of SQL queries which the user inputs. This method of detecting the SQL attacks was tested on web applications that had a heavy interaction with the database. This work employed both manual and the model-based method which was used to test the effectiveness of this approach. The proposed approach was especially beneficial for detecting the web applications. A plus point to this research was its ability to detect the SQL injection attacks that belonged to either code injection, command injection or the file injection.

Bossi et al. (2016) fed light on the insider attacks which were more detrimental to the organizational data than the external attacks. Some internal attestation techniques were to be used to get rid of malicious users, but it might be proved useless because these attestations only run at the start of the application and during the runtime, there was no such mechanism available and also the malicious user could also modify the application for its own use. The proposed system tried to minimize the number of false positives and false negatives and finally, time delays should also be minimized. A new technology for finding anomalies in the database accesses was introduced and called the DetAnom. Forgetting all possible paths this system exercised the concolic testing mechanism that had two kinds of execution one was symbolic execution and the other one was concrete. The queries were built using the signatures and the constraints that were used to issue the query to the database. Hence a novel method was introduced to cope with the challenging anomaly detection scenarios.

George et al. (2016) proposed the concept of reconstruction of queries that resulted due to SQL injection attacks. The basic concept behind this approach was to minimize the denial of the service request and an enhancement in the performance of the database. The basic components of the proposed idea were the SQL query pattern retriever, a template mapper, and a template translator which reconstruct the queries for the sake of eliminating the denial of service situation. The basic concept behind this was to develop the query template for automatically generating the SQL templates. The proposed system detected the multi-variant type of queries such as tautology based injected queries, statement injection, union query, logically incorrect queries, stored procedures, piggybacked queries and alternate encoding queries etc.

Shu et al. (2015) focused on resolving the issue of stealthy attacks that were considered as almost impossible to resolve. The research that had already done in this regard belonged to the legal software attestation and short call sequence verification issues. In this paper, a two-stage algorithm mechanism was introduced to unearth the diverse normal correlation patterns. Most of the anomaly detection mechanisms were divided into two categories, the one was the short call sequence validation and first-order automation transition verification and it used the probabilistic and deterministic verification. The basic approach behind the large-scale execution window was the development of constrained agglomerative algorithms that faces the behavior diversity challenge. A two-stage modeling technique was adopted. In the first stage, the montage anomalies were also detected while in the second stage frequency anomalies were detected.

Xu et al. (2015) proposed the current trends of generating anomalies that were changing nowadays. One example of these stealthy attacks was by using return-oriented programming i.e. normally called as code reuse. The distinction of such attacks was their not using the SQL strings with the normal strings to malign the original query but they were issued by the application programs to the databases. This paper introduced a novel anomaly detection technique that probabilistically modeled and learns a program's control flows for high-precision behavioral reasoning and monitoring. Linux platform was used and the

mechanism was named as STILO. STILO stands for STatically Initialize Markov. New techniques were devised to detect the anomalous behavior that was the new probabilistic control flow model. It extracted the control flow information both statically and dynamically.

Sallam et al. (2017) proposed the DBSAFE architecture to detect the database anomalies that occurred when some insider tried to attempt the ex-filtration of data related to some particular organization. RDBMS was highly vulnerable to the exfiltration of data by the insiders which was really challenging. Some conventional methods were used in this paper to build the profile which guessed the behavior of the application in a normal routine and if any inconsistency was detected in the proceedings, the system immediately ran the predefined policies or alerts. The good thing about the DBSAFE was it's not imposing its constraint on the type of DBMS.

3. Anomaly Detection and Reconstruction Schemas

Most of the researchers proposed an SQL injection detection mechanism to detect SQL anomalies by either employing a static or a dynamic analysis mechanism. The detection mechanism only focused on exploring the anomalies in the queries without bothering about the correction and reconstruction. Hence the recent works were beneficial for the purpose of securing the data against the intruders, but the system definitely responded in the denial of services (DoS), because the actual query passed to the database was blocked by an intrusion detection system. The purpose of this work is to make the system available for the users by correcting and reconstructing the malicious queries before they go to the database for execution. The process works in two phases, the first phase consists of finding the anomalies in the SQL queries while the second phase deals with detecting the anomalies which are to be reconstructed and enable the system to reduce the denial of service state.

The structure of finding the anomalies in the queries is performed by developing a profile, i.e. the signature development and augmenting the required constraints with all the submitted queries [1]. In the dynamic analysis, the signatures of the submitted queries and the constraints are attached during runtime of the application, in that all the hidden paths are continuously explored on a regular basis [4]. The mechanism behind this approach is to issue a query by the application, in the next run, the query is evaluated against the context of the application, i.e. what type of queries can be submitted to the database? So if the query is according to the current context, the query is legitimated otherwise the query is declared as anomalous [5]. The main components of the anomaly detection System are the SQL Receiver, Constraint Extractor, Signature Generator, Profile Binder and Signature Comparator. A centralized module exists between the Anomaly detection module (ADM) and the query reconstruction module (QRM) i.e. the (QDM). The Query Delegation module (QDM) gets its input from the ADM and in the case of legitimate query it transfers the query to the database for execution otherwise it forwards the query to the QRM where the query is evaluated for eliminating the augmented parts. The third component of this system is the query reconstruction module which has the signature parser and anomaly remover for finding the victim query and removes that anomalous part of the query. The anomaly detection module has the central role in this technique as it receives the query from the application and forwards it to either the database or the anomaly reconstruction module. After receiving the query, the ADM interprets the query by making an effective comparison of the signatures along with the query by using a matrix. After an inclusive comparison the query is considered as legitimate in the case when the stored signatures resemble with current query otherwise the query is declared as anomalous if any sort of mismatch is detected [6].

A. Profile Generation Module

The profile generation module fetches an SQL query from the web application and populating the query with signatures and constraints. These signatures and constraints are considered as prerequisites for transferring the query to the database. The profile generation module contains further two submodules, i.e. the signature generation submodule and the constraints generation submodule. Both these sub-modules are combined to form the aggregate profile of the database which contains all the expected queries [7].

B. Signature Generation Sub Module

The signature generation sub-module generates signatures for the query. It actually shows how the query looks like? The signatures prove to be useful for detecting the anomalies in the system. The signature generation is a technical task which can be formulated by generating the codes against different queries such as Select, Insert, Update or Delete [8]. The methodology used to generate the signatures is the Backus Normal form. The Backus normal form is used to encode the SQL syntax in a way that facilitates an easy comparison among different queries. It is an unambiguous meta-language for describing the syntax of other languages [9]. Let’s consider a query.

```
SELECT [DISTINCT] [ATTRIBUTES_LIST]
FROM [TABLES_LIST]
WHERE [QUALIFICATION_LIST];
```

Now the system is going to consider, how the signature generation method converts different SQL query components into the signature codes.

For "SELECT" command, the mnemonic S is used to represent the query. Similarly, U, I and D for the update, insert and delete commands respectively. The proper method for assigning the signature codes is to give some digit value that may start from some discreet value like '301' or '401' or '501' etc. if the value 300 is given to a table, its columns should start from 301 and consecutively assigns values.

```
SELECT pin_code, user_name, password
FROM tbl_users
WHERE id=243;
```

The signature for the above SQL string is as under.

```
(S, {302,303,304}, {300}, {301}, 1)
```

In the same way, the signatures of other queries like Insert, Update, and Delete commands can also be designed e.g.

```
INSERT INTO {Table Name}
{Attributes List}
VALUES {Attributes Values}
```

The query looks like this.

```
INSERT INTO tbl_employees
(pin_code, User_name, Password)
VALUES (244533, 'SohaibYousaf', 'abc@123')
```

Here in the case of an insert command, only a single relation is being used. The column names are also displayed proceeding by the values that are assigned to those columns respectively. So the signatures for an insert command may look like.

```
(I, {COLUMN NAMES}, {TABLE NAME}, ∅, 0)
```

4. The Concolic Execution

The concolic execution is used to explore the execution paths of an application. It works by taking the application as input to the concolic execution engine. The concolic execution works by traversing all the execution paths of an application. The traversal of execution paths is done purely for creating a database of all the possible execution path flows. The execution engine works by following the branch conditions by taking the concrete inputs initially and fetches the other execution paths subsequently. The concolic execution takes constraints solver to reverse the branch conditions [10]. These reverse branch conditions enable to explore more execution paths and this execution is repeated for a number of times, so this way almost all the execution paths are traversed and the database is populated with execution paths. The core of concolic execution is the profile generation phase in which the constraints are added to the query and finally the signatures are extracted [11].

5. Profile Creation

Profile creation phase demonstrates how the query records are built by the profile builder? How are these query records arranged to form a profile of the application? The application profile can be considered as a directed tree which is denoted by P. The relation between the profile P the tree T can be described as T (V_P, E_P). The nodes of a tree can be represented by V_i and $V_i \in V_p$ which is a query record of query q_i and it can be represented as <sig(qi),ci>. Here in this equation sig (q_i) represent the signature of qi. C_i in the equation represents the constraints used to execute the query in the database. The edge $e_{ij} \in E_P$ which denotes the actual query qj which is executed just after the query q_i and the node v_j behaves as a child of node v_i. The concolic execution traverses the branch conditions in a well-defined instrumented environment.

Example: Concolic Execution Path Finder

```
1- Private void PriceManagement (int decrease_amount,
item_id, increase_amount){
2-Statement s;
3-int sales_count = 0;
4-String query_1 = "SELECT SUM (sales_quantity) as
Total_Sales FROM tbl_items WHERE item_no=item_id;
5-result_Set1 = s.executeNonQuery (query_1);
6-result_Set1.last();
7-if (Total_Sales < =10000) {
8-String query_2 =SELECT Item_Size as Product_Size
FROM tbl_items WHERE item_no=item_id AND
sales_price>=500;
9-result_Set2= s.executeNonQuery (query_2);
10-if(result_Set2.getRow()>=100){
11-String query_3 = "UPDATE tbl_items SET item_price
= item_price - decrease_amount WHERE
item_no=item_id AND Item_Size= Product_Size ";
12-result_Set3= s.executeNonQuery(query_3);
13- }
14- else {
15-//Do some other operations
16-}
17-else {
18-String query_4 = "UPDATE tbl_items SET item_price
= item_price + increase_amount WHERE
item_no=item_id ";
19-result_Set4= s.executeNonQuery(query_4);
20-} Return 0;
21-}
```

The theory behind this concolic execution is the input that it takes in the form of an intermediate language such as a Java bytecode. The Java bytecode is used to find the branch conditions and also to execute it in an instrumented environment. The program shown above describes the price control system of the commodities to increase the sale.

Table 1: Constraints In Profile Creation

Constraint	Type	Condition
C_1	Database	X1<=10000.0
C_2	Database	X2 >= 100.0
C_3	Database	X2 <100.0
C_4	Database	X1 > 10000.0

Table 2: Query Signatures

Query	Type	Signature
Query_1	Select	{S,{201},{200},{202}, 1}
Query_2	Select	{S, {203},{200},{202}{204} ,2}
Query_3	Update	{U,{200},{204},{202},{205},2}
Query_4	Update	{U,{ 200},{204}, {202}, 1}

The instrumented environment means to execute the branch conditions in an order without getting the dynamic requests by blocking these requests and passing the bytecode automatically generated values by the instrumented environment. To implement this, concolic execution passes the values in a way so that all the branch conditions should be executed one by one and no path should be left as un-traversed [12]. In the above example of the program, the price control mechanism uses a “0” value that is passed to the “price management” function as a parameter to prove the first condition as a true.

In the first condition i.e. 0<=10000 proves true and the first query is executed that means there is less sale than expected figure and the price of the costly commodities needs to be lessened. The constraints imposed by these statements are shown in the constraints table as c_1; similarly, the query is also depicted in the queries table. The methodology behind executing this particular bytecode is that the query does not reach the real database but they are blocked by the environment instrumentation, rather than executing it by the database [13]. Hence all the signatures, constraints, signature profiles are shown in their respective tables as shown below. On the other hand, the complete profiles are shown in the Query Profile figure below.

Table 3: Query Profile Signature

Query	Profile Signature
PQ_1	<sig(query_1),c_1>
PQ_2	<sig(query_2),c_2>
PQ_3	<sig(query_3),c_3>
PQ_4	<sig(query_4),c_4>

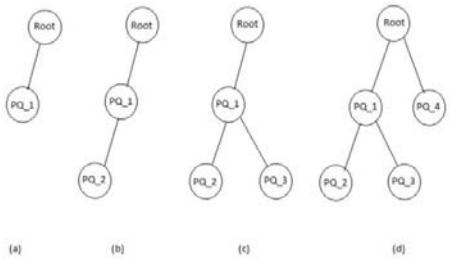

Fig. 1 Query Profiles

6. Anomalies Detection Module

In the anomaly detection phase, it is decided which queries are considered as legitimate and which are not? There are various mechanisms to detect the anomalous queries [14]. One of the two methods is discussed in this work, i.e. comparison can be made by either of the two methods. One is named as linear algorithm while the other one is the Hirschberg algorithm, however, the linear algorithm is followed in this work.

6.1 Anomaly Detection Algorithm

In the first phase, linear algorithm is discussed for detecting the SQL anomalies. The algorithm is as under:

1:Start
2:Input: Branch Conditions in Application Profile
3: Nr = root node
4: while program executes
5: Qi = issued query
6: Ci = input constraints
7: The signature generator generates required signature sig(q)
8: Flag = false
9: for each child Ni of Nr
10: if Ci satisfied
11: signature comparator compares profile signatures sig(q) to query signatures sig(queryi)
12: if matches
13: set Legal Query
14: Nr = Ni
15: else
16: set Anomalous Query
17: end if
18: flag = true
19: break
20: end if
21: end for
22: if flag == false and Nr is an incomplete node then
23: set WARNING
24: end if
25: end while
26: End

The detection of the anomalous queries is performed by passing the queries to the query delegation module which forwards the query to the anomaly detection module rather than directly to the database.

7. Query Reconstruction Module

The query reconstruction module (QRM) takes the anomalous query as input and makes an effective comparison of the number of columns and the number of predicates of the stored query in the query profile. As it is already known, the numbers of predicates are stored in the profiles of the query so it will become very easy to distinguish the input query from the already stored query in the query profiles. Here one thing to mention is the system's completeness because the system proves effective only in the case when the profiles are developed in a complete and comprehensive manner otherwise the system will only manage to produce the warning messages that are no more constructive and reliable because sometimes during the profile creation phase the depth bound search is reached before the profile is completely generated. The comparison of the input query with the already stored query is quite simple as

```
String query_1 = "SELECT SUM (sales_quantity) as Total_Sales FROM tbl_items WHERE item_no=item_id;
2-result_Set1 = s.executeNonQuery(query_1);
3-result_Set1.last();
4-if (Total_Sales < =10000) {
```

Table 4: Constraints

Constraint	Type	Condition
C_1	Database	X1<=10000.0

Table 5: Database Constraints

Constraint	Type	Condition
C_2	Database	X2 >= 100.0
C_3	Database	X2 < 100.0
C_4	Database	X1 > 10000.0

Table 6: Query Signatures

Query	Type	Signature
Query_1	Select	{S,{201},{200},{202}, 1}
Query_2	Select	{S, {203},{200},{202} {204} ,2}
Query_3	Update	{U,{200},{204},{202},{205},2}
Query_4	Update	{U,{ 200},{204}, {202}, 1}

Table 7: Queries Comparison

Sr. No		0	1	D1	2	D2	3	D3	4	D4
1	Query Type	{S}	{S}	00	{S}	00	{U}	Ø	{U}	Ø
2	Columns	{201}	{201}	00	{203}	01	{200}	Ø	{200}	Ø
3	Tables	{200}	{200}	00	{200}	00	{204}	Ø	{204}	Ø
4	Predicates	{202}, {400}	{202}	01	{202},{ 204}	01	{202},{ 205}	Ø	{202}	Ø
5	Predicates Count	{2}	{1}	01	{2}	00	{2}	Ø	{1}	Ø
6	The difference with predicates count			02		02		Ø		Ø
7	Difference without predicates count			01		02		Ø		Ø
8	Net Difference (6+7)			03		04		Ø		Ø

The query in the constraints table belongs to the select statement and the constraints are checked in the constraints table to ensure the correctness of the query. Two simple options are available i.e. either the Total_Sales is lesser than 10000.0 or it is greater than 10000.0

The queries in the database constraints table prove true and result_Set1 returns records that are less than 10000.0. The constraint C_1 is satisfied and the query is forwarded to the Signature generator module. Let's say the constraints in the query signatures table results in something illegal from the constraints defined, then the query is illegitimate here but it will rarely happen because the constraints usually display a complete range of values that surely come otherwise. In such situations, the query is forwarded to the signature generator module. In this case, a queries table is used instead.

The query in this example matches to the Query_1 signature. Because one knows that only single column is used in this query which is (sales_quantity) as Total_Sales which is denoted as {201} in the signature i.e. {S, {201}, {200}, {202}, 1}

The table name is denoted by {202} in the signatures that denote the 'tbl_items' in the items table. And the number of predicates is also defined unambiguously in the signatures, i.e. a '1'. In the case of this query

```
item_no = item_id;
```

The item number is compared to the number stored in the database. In this way, the query is legitimated in an effective manner. In another case where the query is not considered as legitimate and an illegal query is issued by the application, the query format may correspond to the following.

String query_1 = "SELECT SUM (sales_quantity) as Total_Sales FROM tbl_items WHERE item_no=item_id and '1'='1';

7. Extraction of Victim Query

In the query reconstruction phase the thing that is clearly understood is the problem in the received query usually somewhere in the syntax of the query, so in the reconstruction, the most important thing which is to be identified is the identification of which query is actually augmented with a malicious code. There is a need for a comprehensive mechanism to detect the affected query.

For this purpose, a table is drawn by the Anomaly Reconstruction module.

The mechanism behind the reconstruction of the queries is depicted in a tabular form as the first field of the table shows the serial No. The characteristics of the queries are shown in the second field of the table. The third column '0' represents the anomalous query which is issued by an intruder. The query properties and predicates are separated in different tuples of the table. The next fields are '1', '2', '3' and '4' represents the query signatures which are already stored in the database. The subsequent columns contain the differences between the anomalous query and the signatures respectively. Here in the query described above belongs to the "Select" type so a comparison can only be made among the signatures of the same selected type while the update signatures are discarded and shown by a 'Ø' in the table, hence no difference can be calculated of these types. After that, the difference of the anomalous queries is calculated with all the stored signatures. The final step at this point is to calculate the net difference which is analyzed further to reach a final decision. Now it can be seen that the net difference is minimum at the query_1 so query_1 is assumed to be the query which is maligned by the outsider. Now the procedure got simpler in a sense as it becomes very easy to differentiate the augmented parts from the actual query.

A. Victim Query Detection Algorithm

The anomaly reconstruction algorithm is devised to alter the structure of an already catch query so that the query is reconstructed and forwarded to the delegation module for submission to the database:

1: Start
2: Input: Query to be Reconstructed
3: Qp = Array of Profile Query
4: Qa = Array to store the Query
5: Qi = Array to Store input Query
6: Dw = Difference with Predicate Count
7: Do = Difference without Predicates Count
8: Di = Array to store the Difference
9: Dn = Net Difference
10: While program executes
00: While true
11: if Qi equals Qp
12: Difference is stored in Di
13: End if
14: End While
End While
15: for each index Dn of Dw
16: Difference without Predicates count is stored in Do
17: Difference with Predicates count is stored in Dw
18: Dn = Do + Dw
19: End for
20: Min = Dn
21: For each index of Dn
22: if Min is smaller than Dn
23: Min=Dn
24: End If
25: End for
26: Return
27: End

B. Query Reconstruction

The query reconstruction involves, establishing an application that took the malicious query and eliminates the augmented parts. In this work, it is performed by using a matrix in which several columns are used to produce the original query [15].

The first and second column in the table: VIII represents the serial number and the properties of the query respectively, while the rest of the columns include the original query having column number '0', the signature column having column '1', and the difference column having the heading 'D1'. The 'Difference' column shows the calculated results by subtracting the predicates in the anomalous query from the signatures column and the last column contains the reconstructed query i.e. the actual query. Hence the column with the 'Actual Query' heading contains the desired query.

Table 8: Query Reconstruction

Sr. No		0	1	D1	Difference	Actual Query
1	Query Type	{S}	{S}	00	00	{S}
2	Columns	{201}	{201}	00	00	{201}
3	Tables	{200}	{200}	00	00	{200}
4	Predicates	{202}, {400}	{202}	01	{400}	{202}
5	Predicates Count	{2}	{1}	01	{01}	{01}

So the actual query formed is the {S, {201}, {200}, {202}, 01};

The actual query is reconstructed by the reconstruction module by setting the constraints into actual column names e.g.

```
String query_3 = "UPDATE tbl_items SET item_price = item_price - decrease_amount WHERE item_no=item_id AND Item_Size= Product_Size ";
```

C. Query Reconstruction Algorithm

As when the victim query is found, the next task is to reconstruct the actual query so that the query should be forwarded to the database for the execution of the query to minimize the denial of service state. The algorithm for the reconstruction of the actual query is written below.

```
1: Start
2: Input: Victim Query
3: Qp = Array of Profile Query
4: Qs = Array to store the Query
5: Qv = Array to Store victim Query
6: Di = Array to store the Difference
7: Da = Array to store the actual query
9: While true
10: if Qv equals Qp
11: Difference is stored in Di
12: End if
14: End While
15: for each index of Di
16: if value of Di equals 0 then
17: Da = Di
18: Else
19: Da=Di – Dv
20: End if
21: End for
22: Return
23: End
```

9. Query Reconstruction Implementation

The next phase describes how the reconstruction of queries is performed in this system.

A. Finding the Victim Query

This is the first step in the reconstruction of the queries. Here a matrix is made to extract the differences among the queries. The matrix forms as the ‘0’ field of the matrix contain the anomalous query while the next ‘1’ field of the matrix contains the other query signatures that have already been stored in the database so consequently every field next to the stored signature contains the difference field of every query. The difference count of the queries is also maintained in the net difference field of the query. The query with the minimum net difference is declared as a maligned query.

B. Extracting the Anomalous Parts

The original query extraction phase utilizes the same mechanism as in the case of finding the victim query by utilizing the matrix for extracting the differences between the anomalous query and the stored signatures in the database. The work follows as the field ‘0’ contains the augmented query while the field ‘1’ contains the stored query signatures so a difference is calculated at the field ‘D1’ and the augmented parts in the field “Difference”. Subsequently, the actual query is positioned as the last field of the matrix which is in-fact the same field as the field ‘1’.

C. Rebuilding the Original Query

The original query is reconstructed by developing a matrix as in the case of finding the victim query. The difference between the anomalous query and the query signature forms the actual query. The whole work is done to eliminate the denial of service state which might come when an anomaly is detected in the query. After the reconstruction, the query is forwarded to the database for execution and as a result, a response is sent to the computer from where that query was originated. But in a case when the query has not constructed the control backtracks to the anomaly detection module where the system just generated a warning message that the query is anomalous and the system is unable to reconstruct it.

10. Experimental Evaluation

The workflows as the complete system are to be developed where both the modules i.e. the anomaly detection module and anomaly reconstruction module are transformed into two different servers or a single server with two modules. These servers need to be installed in such a manner that one server or module receives the query from the client computer and it just checks the legitimacy of the query. In the case when the query is declared as a legal query it should forwards the query directly to the database where the query should be executed without any further delay but if the query is declared as anomalous, the query is diverted towards the anomaly reconstruction server or module for subtracting the anomalous parts from the query and when the query is declared as purified, the query is transferred to the database to eliminate the denial of service.

11. Threats to Validity

The created application has a thorough mechanism of finding database anomalies but certain points are there

which should be considered while assuring the validity of the application e.g. owing to the volume of the application, instrumentation of all the instances of a big application may not always be possible. In some situations, time constraints may come intact because this application may install on a large number of computers and update all those computers may not be an easy task to do. There are some technical issues as well, such as some environments completely deny the exposing of corresponding API's for any type of intermediate code or languages. Sometimes the performance may also be affected because a large number of computers are connected to the application which may create performance overhead. The time constraints in this framework are quite a complicated issue because the profile generation is a time-consuming task because it has to traverse all the execution paths of the system. The importance of false positives and false negatives also proves to be an obligation for the system because it only creates overhead for an already crowded system. Sometimes it may happen that a wrong query will be selected as a victim query, but it does not happen very often because the query, in this case, carries the complete predicates, this is why the chance of selecting a wrong query as the victim query has a minimum probability. This can be a case where the query is dissected and several of its parts are eliminated from the actual query.

12. Conclusion and Future Work

The system is developed with the intent to abstain the attacker from executing the queries which are anomalous. The reconstruction module performs correction tasks by eliminating the malicious parts of the query. The mechanism provides a dynamic analysis that creates constraints and signatures and stores that information in the profiles [16]. The proposed work also makes some arrangements for reconstructing the SQL queries that contain the SQL injections. A brief comparison of the queries is made to dissect those parts from the actual query that are injected by some malicious user for the intent of making a loss to the individual or company [17]. The responsibility of the system is to minimize the response time by eliminating the denial of service situation.

For managing online applications, there is a need to systematize the development and management process because online applications are more vulnerable to security threats. In dynamism, the next goal of this work will be to enhance the signature generation by providing information about the constants, variables in the where clause of the query. Similarly, better models and algorithms can be devised to reconstruct the queries. There is a need to develop a process by which the queries are not re-tested that has already been tested. Similarly, there is a need to lessen the time depletion while doing the detection and correction work. A comprehensive plan can be developed which will automate the reconstruction process with better utilization of the parameters.

References

[1] Bossi, L., Bertino, E., & Hussain, S. R. (2017). A System for Profiling and Monitoring Database Access Patterns by Application Programs for Anomaly Detection. IEEE Transactions on Software Engineering, 43(5), 415-431.

[2] Bertino, E., & Ghinita, G. (2011). Towards mechanisms for detection and prevention of data exfiltration by insiders: keynote talk paper. In Procou8eedings of the 6th ACM Symposium on Information, Computer and Communications Security (pp. 10-19). ACM.10-19.ISBN: 978-1-4503-0564-8.

[3] Valeur, F., Mutz, D., & Vigna, G. (2005). A learning-based approach to the detection of SQL attacks. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (pp. 123-140). Springer, Berlin, Heidelberg.

[4] Tajpour, A., Ibrahim, S., & Masrom, M. (2011). SQL injection detection and prevention techniques. International Journal of Advancements in Computing Technology, 3(7), 82-91.

[5] Garcia-Font, V., Garrigues, C., & Rifà-Pous, H. (2016). A comparative study of anomaly detection techniques for smart city wireless sensor networks. International Journal of Sensors, 16(6), 868.

[6] Patcha, A., & Park, J. M. (2007). An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer networks, 51(12), 3448-3470.

[7] Lee, I., Jeong, S., Yeo, S., & Moon, J. (2012). A novel method for SQL injection attack detection based on removing SQL query attribute values. Mathematical and Computer Modelling, 55(1-2), 58-68.

[8] Som, S., Sinha, S., & Kataria, R. (2016). Study on sql injection attacks: Mode detection and prevention. International Journal of Engineering Applied Sciences and Technology, Indexed in Google Scholar, ISI etc., Impact Factor: 1.494, 1(8), 23-29.

[9] Lee, K. D. (2008). Programming languages: An active learning approach. International journal of Springer Science and Business Media.

[10] Sen, K. (2007). Concolic testing. In Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering (pp. 571-572). ACM.

[11] Majumdar, R., & Sen, K. (2007). Hybrid concolic testing. In Proceedings of the 29th international conference on Software Engineering (pp. 416-426). IEEE Computer Society.

[12] Kapus, T., & Cadar, C. (2017). Automatic testing of symbolic execution engines via program generation and differential testing. In Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (pp. 590-600). IEEE Press.

[13] Jensen, C. S., Prasad, M. R., & Møller, A. (2013). Automated testing with targeted event sequence generation. In Proceedings of the 2013 International Symposium on Software Testing and Analysis (pp. 67-77). ACM.

[14] Ciampa, A., Visaggio, C. A., & Di Penta, M. (2010, May). A heuristic-based approach for detecting SQL-injection vulnerabilities in Web applications. In Proceedings of the

2010 ICSE Workshop on Software Engineering for Secure Systems (pp. 43-49). ACM.

[15] Sadotra, P. (2015). Hashing Technique-SQL Injection Attack Detection & Prevention. International Journal of Innovative Research in Computer and Communication Engineering, 3(5), 4356-4365.

[16] Alwan, Z. S., & Younis, M. F. (2017). Detection and Prevention of SQL Injection Attack: A Survey. International Journal of Computer Science and Mobile Computing. 6(8). 5-17.

[17] Kindy, D. A., & Pathan, A. S. K. (2011). A survey on SQL injection: Vulnerabilities, attacks, and prevention techniques. In Consumer Electronics (ISCE), 2011 IEEE 15th International Symposium on (pp. 468-471). IEEE.

[18] Dalai, A. K., & Jena, S. K. (2017). Neutralizing SQL Injection Attack Using Server Side Code Modification in Web Applications. Security and Communication Networks, 2017.

[19] George, T. K., & Jacob, P. (2016). A Proposed Architecture for Query Anomaly Detection and Prevention against SQL Injection Attacks. International Journal of Computer Applications, 137(7), 11-14.

[20] Shu, X., Yao, D., & Ramakrishnan, N. (2015). Unearthing stealthy program attacks buried in extremely long execution paths. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (pp. 401-413). ACM.

[21] Xu, K., Yao, D. D., Ryder, B. G., & Tian, K. (2015). Probabilistic program modeling for high-precision anomaly classification. In Computer Security Foundations Symposium (CSF), 2015 IEEE 28th (pp. 497-511). IEEE.

[22] Sallam, A., Bertino, E., Hussain, S. R., Landers, D., Lefler, R. M., & Steiner, D. (2017). DBSAFE—an anomaly detection system to protect databases from exfiltration attempts. IEEE Systems Journal, 11(2), 483-493.