# An innovative model approach to prevent malformed pro sql tuples in general DB model

G. Subhasree
*Department of Information technology*
*VR Siddhartha engineering college, Vijayawada, Andhra Pradesh, India*

Dr. G. Ramakoteswara rao
*Department of Information technology*
*VR Siddhartha engineering college, Vijayawada, Andhra Pradesh, India*
*Email- grkraoganga@gmail.com*

Dr. P.Vidya Sagar
*Department of Information technology*
*VR Siddhartha engineering college, Vijayawada, Andhra Pradesh, India*

**Abstract- SQL injection is a way that directs to attackers to attack the database in non-sign in way and reach sensitive data directly to extract and malfunction. In this work we investigate various attacks elaborated practical scenarios and how the attackers attacks and also how to prevention of these kind of attack modes in web (W3C) and non-web models. Our work also concerns process and cons of merits and demerits of SQL injection attacks. It is well know that SQL injection prevention can be easily traced and prevented by utilizing more secured approaches at pre and post user login modes. Our work is extended to implement this prevention called LR and RL approaches These two ways may be weak in login mode attack prevention but very effective in preventing the query level so that sensitive data leak can be restricted.**

**Keywords – SQL injection, prevention, query, database, web, W3C.**

## I. INTRODUCTION

SQL injection is most well-known issue to effect the prominent and huge business and leads to leakage of huge sensitive data credentials, secret identifications, transaction details, personal information. In our daily W3C usage (E Commerce, IT portals, commodities) we connect for surfing with transactions, normally to login and populate after login portals will connect to huge databases. So all these kind of process is based on data-driven.

These databases contains all the information which is user based and population of data depends on the code written in web server side by some languages but embed with SQL queries with inner tuples. So at this stage SQL injectors can inject malformed information to populate the sensitive data which is transacted data from database. Almost all web based portals are totally database driven now a days [4]. E Commerce applications are vast database driven web applications in our daily transactions. At the other end of the coin for these portal's corporates will use low cost software to maintain the data administration levels. In this mode there is nothing less impact that the information is so precious in business level and gaining peak level of security can be viewed as imperative in the process of maintenance of competitive model. But SQL injection attacks are most dangerous and severe issues in web applications level. They are frequently attacked by attackers for various reasons like financial frauds, getting the sensitive data, degradation of the popular websites. The frequent number of attacks for the popular websites drastically getting increased, leads to scaling of attacks are more in W3C. By considering these kind of attacks the SQL injection prevention topic has become interesting and trendy in research criteria. Lot of approaches

with respect to specific attacks to develop the counters, but most of them cannot match the accuracy in pretending attackers and injections in web application level databases. One prominent approach is to avoid attacks and injections to frame batch level SQL queries like procedures to populate data from databases.

## II. SQL INJECTION

This is a type of threat to huge web application's security in which attackers may inject or submit SQL query to populate sensitive data and to expose the database. Most of the time attackers take advantage at user signing up level. At sign up if the program on web server is not maintaining and validation or proper security methodologies which are the part of processing the query. The smartly framed SQL commands leads to padding the data or changing the process of execution level to change the data population. This SQL Injection allows attackers to create, read, update, alter, or delete the data stored in relevant database [10]. In most common cases SQL Injection which allows attackers to control the sensitive data like personal information, transactional passwords and other financial related data. According to global survey SQL Injection is most dangerous threat to web level security.

Conceptual levels of SQL Injection:

- This is programmatically threat that raises when user enters data and that data interprets by SQL query which is a part of regular SQL query management.
- Significant moulded data will be embedded into SQL queries by attackers and manages to execute those commands.
- Attackers uses this threat by padding this significant moulded input and injected data in such a way that query management and processing interpreters cannot be able to distinguish those moulded data from original one [9].

These types of threats process at database layer level. By executing specific commands (SQL queries) attackers can update, delete, and modify the data in the existing database.

## III. TYPES OF SQL

Normal database applications became key component model in control systems and their related tuple maintenance methodologies. Regular security tries to protect systems by isolating existing software components and targeting security efforts opposite to threats to particular computer developed software component methodology.

### *A. Conditionality*

The main aim of conditionality-based Attack is to pad the extra code in one or more conditional SQL statements that always executed and processed as true output. The outcome results of this attack depends how this attack uses with in our regular framework of application. The prominent usage are to reroute evolution and authentication process and extra information [13]. In conditionality attacker exposes padding table that is used in "WHERE" clause based SQL query. Converting this query into conditionality leads to all tuples targeted by SQL query to be returned.

*Example*

In this sample attack the attacker promotes [or e=e – "] for credentials as username as input (not taking other conditions which are submitted as inputs) which leads to following:

```
SELECT address FROM user_tbl WHERE
name = " " OR 9 = 9 AND pwd = ' '
```

The padded information in the condition [OR 9 = 9] converts the entire WHERE conditional clause into a conditionality. The main storage database uses the conditional with respect each tuple has to be returned to main framework. The main reason behind this is always raised query returns "TRUE" for all tuples [6].

### *B. Wrong/Incorrect logical framing*

These kind of attacks leads to attackers to popup sensitive information about the model and kind of structure of the main stored database which is related to web infrastructure. This kind of attack is preauthorized data capturing steps to attacks. Basically application servers gives default error pages with respect to regular web errors returned by servers' error table. These kind of error output, basically tries to developers to clear the issues and attackers takes advantages to gain information schema based on backend database.

*Example*

This kind of intension to inject the code as to reveal relevant data. To get this attackers injects the following un-formatted data leads to following URL

```
(qeury) UNION SELECT <desired_column> FROM
DATA _SCHEME.columns WHERE tbl_name = "uname "_
```

```
www.<domain_name>/domain-users.aspx?uid = "xyz"
```

The final incorrect SQL framed query is

```
www.<domain_name>/domain-users.aspx?uid = "xyz"
UNION SELECT <desired_column> FROM DATA _SCHEME.columns
WHERE tbl_name = "uname "_
```

The enhanced and investigated infrastructure contains the scenarios, which will guard the running web application from all above SQL padded attacks.

*C. UNION based query*

By this scenario, attackers join based padded query to the protection mode query to the protected by the UNION word, then can populate information from other non-related tables from regular web application.

*Example*

Here is the processed from the application based webserver.

```
SELECT <col_name1> ,<col_name2> FROM
    usr_tbl WHERE col_uid=*id
```

By padding the following user id value

```
*id = 6 UNION ALL SELECT SOCIALSNUMBER, 7 FROM CC_TBL
```

The resultant query:

```
SELECT <col_name1> ,<col_name2> FROM
    usr_tbl WHERE col_uid = 6 UNION ALL
SELECT SOCIALSNUMBER, 7 FROM CC_TBL
```

*D. Results of SQL padding(injection)*

All the attacks shows effects on database which is related to web application and current scenario where the data storage is one major prioritized origin incorporates/organizations. So with these injections all attackers can take complete grip on the database, can able to do malfunction with the databases.

Here are the following things can happen with attackers:

1. Regular sql INSERT command to populate all the user account information in a system, which is totally sensitive like (usernames and passwords).
2. Deletion of sensitive contents in the tuples.
3. Insert invalid user account information.
4. Stopping the start-up service of database engine.
5. E-commerce change of product details for free of cost.
6. Invalid file additions.
7. IP addresses track.
8. Malfunction of files which are relevant to OS.

All the database injection attacks leads to financial down and loss. This leads to drastic damage on applications with respect incremental growth of corporates/organizations. So end users stops using the applications, due to the fear of their valuable information reveal through the application [2].

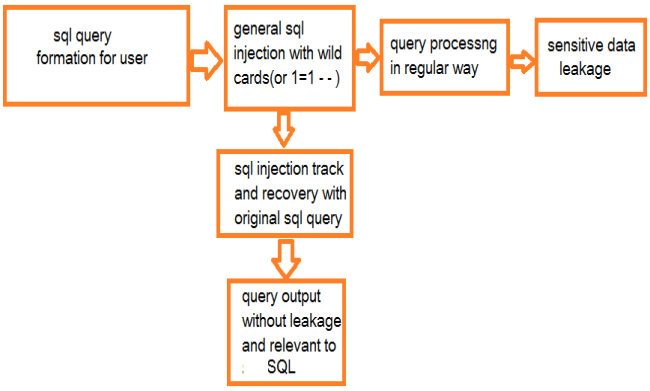


Figure 1. Architecture of SQL injection

## IV. PROBLEM STATEMENT

Due to growth of various crawling websites which are mounted on numerous webservers, which provide certain features and services that are part of the relevant website and uses third party software to manage their data. These applications not only accessed by private persons but also by reputed organizations and government related persons. Mainly the database is used as a prioritized source to access the user relevant requested data. Somebody will store the data which related to some elements and relevant to websites or information created by internal website departments. The number of security based issues raised in this scenarios of rapid and dynamic webpages to their users of website. The attacks exposes the security information and sensitive information existing on database like credit card numbers and social security numbers too [15]. This is possible to populate information is only possible with sql injection/padding, the reason is framed SQL query (which is already effected by attackers) will not be checked for correction but processed. Which are also by passing the login levels. Which can lead to install third party software to be installed on web applications and servers too.

Sql injection/padding is a serious threat and exposes application level layer vulnerabilities in W3C applications. Rather attacking objects like web servers or operating systems, the use of SQL injection/padding all databases, servicing as database servers through web applications (portals). There are many scaling that can be taken to suppress SQL queries. There are numerous proposals in normalizing SQL-attacks. TONSIA (fig 1) checks the SQL query in steps. The first step recognizes pushers which are used to pump SQL queries to database. Second it frames automation model regular attacks in the form of non-estimated finite automation. Finally its checks for the malicious and exposes outside. This really comes to execution level regular tracking analysis and monitoring database engines [18]. In static level part scenarios will be used programmatic analysis to automation building of model level SQL queries which generated by web portals. In dynamic model level the scenarios monitors the instant generated queries tracks them for acceptance which is output of statistically generated.

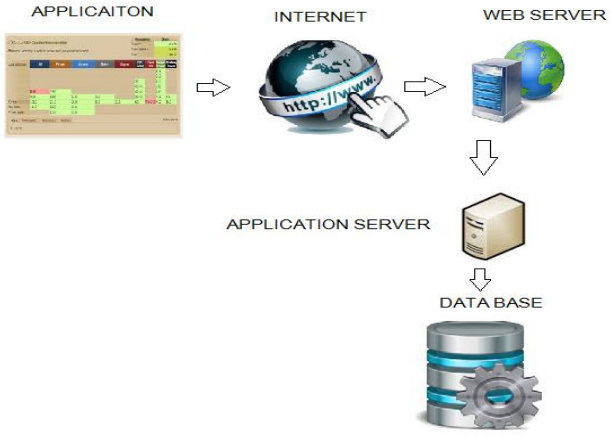


Figure 2. TOSNIA

After SQL injection/padding approaches, our work proposes a new technique in which our work uses to implement a mechanism, detects and vanishes the threats SQL injections/padding technique of Novel and encryption

models [11]. The concept is very simple based on novel and encryption. Rather than user's boundaries issued by admin level our work implements the smart coding methodology in defensive model to secure our web portals.

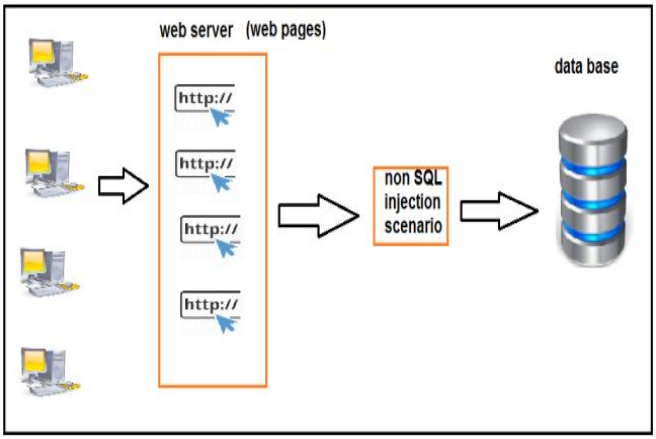


Figure 3. Prooposed approach of SQLIA

Proposed technique works in a simple way on novel model for secure signup technique. In which we compute an unique value for all users (unique) with respect to credentials and saves as views in databases as simple username/passwords.

| ID | UNAME | PASSWORD |
|---|---|---|
| 1 | uname1 | un123 |
| 2 | dileep | Dil123 |

Table 1. User table

| Novel UNAME | Novel PASSWORD | NOVEL EX_OR |
|---|---|---|
| 4nom38nknasd | Ndqwenbaisdf | 489dnmdnlmm |
| 8ndknqekmmd | Jdmduomj9 | Ddccemen |

Table 2. Generating hash values

| Normal SQL query | SELECT uname , pword FROM <tbl> WHERE uname = #unam AND pword = #pwd |
|---|---|
| Effected query | SELECT uname , pword FROM <tbl> WHERE uname ' ' OR 5 = 5; '/% AND pword = ; '/% |
| SQL injection/padding | SELECT uname , pword FROM <tbl> WHERE NOVELEX_OR = EXOR(NOVELVAL('$user mm') , NOVELVAL('$pwd')) |
| Output | NOT POSIBLE , COZ ORIGINAL VALUES NOT GOING TO sql Query |

Table 3. Hashed table

*Hash function algorithm*

With this proposed technique frame work need an extra column in database with regular inclusion of tables. Whereas EX_OR of the novel values of user's credentials at the time, where all users registers for the first time and preserves in database. When in signup process identification will be evaluated with existing credentials with novel value. These novel values are framed when user tries to signing up process.

During the authentication of user, the SQL query with novel attributes will be considered and used too. So when user singing up process proposed scenario working with SQL query, automation in detection with the malicious contents and stops the process of getting the values.

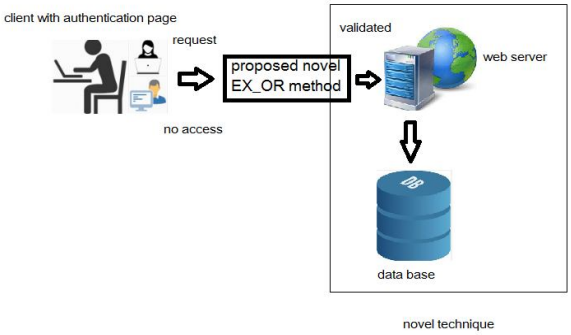


Figure 4. Hash function for preventing SQLIA

The reason of not by passing the validation process. The advantage our proposed work is that attacker do not aware of about novel values of credentials [12]. Which can leads to the attacker not to bypass the authentication process with respect general SQL injection/padding scenarios. The injection/padding is possible on codes which are padded with user registration form but the novel values are calculated at working time at database level results attacker cannot estimate or calculate the novel values as it is dynamic at run/working time.

## VII. IMPLEMENTATION AND RESULTS

Our proposed work is totally implemented on a web portal using glassfish server3.44 and MySQL 5.7.120 and tested on windows environment. The proposed work using novel & encryption methodologies works on following modes:

### *A. Simple mode*

In this mode frame work works on totally non secured environment where SQL injections/attacks are not ignored or tracked because user inputs data is directly driven to database manager for processing. So this leads to conclusion that there is no mechanism in this mode to detect the padded code or SQL query & prevent from execution.

| | |
|---|---|
| **Regular query** | **SELECT * FROM <tbl> WHERE uname = '#usrname' AND pwd = '#pword'** |
| **Attacked code** | **'#usrname = ' OR 6 = 6 pwd = '#pword'** |
| **SQL Injection** | **SELECT * FROM <tbl> WHERE name = ' ' ' OR 6 = 6 pwd = '#pword'** |
| **Output** | **All tuples as totally condition is TRUE** |

Table 4. Regular query attacked with injected query

In simple mode of proposed one no security from authenticated data population is driven. Hence even after validation someone can do bypass.

| | |
|---|---|
| **Standard query** | **SELECT uname , cdetails from <tbl> WHERE uid=#id** |
| **Padded code** | **#uname =7 UNION ALL SELECT ssno, 4 FROM <tbl>** |
| **SQL Injection** | **SELECT uname , cdetails from <tbl> WHERE uid= 7 UNION ALL SELECT ssno, 4 FROM <tbl>** |
| **Output** | **YES joins the result of the query with all SS numbers** |

Table 5. Standard query with padded code

In this mode where user can alter the root SQL query by routing UNION operator with the padded code in boxes. This leads the users to populate the data, which has boundaries for general user population.

*B.Protected mode*

In this mode our work took step against to SQL injections/padding by forcing with some rules and mechanisms. Which alkalizes the users to pushed data to make instant decisions whether it is padded or not. If found padded code, simply ignores the database access input query.

Our work is totally protects following attacks:

- Conditionality
- Wrong / Query with incorrect logical framing
- UNION based query

For conditionality attacks our proposed work has the key feature of using EX_OR for novel values for credentials which shown in table3.

To clear Wrong / Query with incorrect logical framing to track and guard the structure of the query as well as data pushing through the argument and returned 1/0 values 1 means LEGIMATE request and 0 means MALFRAMED REQUEST therefore on the basis of rule or decision [1]. All error messages associated with to developer defined issues which leads to secure the application database server from leak of sensitive schema description.

*Example*

Let's consider the inputted value "123", SQL query framed will be

```
select * FROM <tbl> wher uname = '123' and pword = '';
```

Associated error message

```
select * FROM <tbl> wher uname = '123' and pword = '';
```

So from the above discussion framed error message will be

```
Incorrect uname / pword
```

Last one UNION based query injections and to manage these attacks we took the concept on encryption mechanism. We take the help of encryption key in our database and populate the user credentials at the required time by with already associated simple credentials.

REFERENCES


[1]. Dries Buytaert. Drupal.org Community plumbing. http://drupal.org/, 2006.

[2]. CERT Coordination enter. CERT/CC Statistics 1988-2005.http://www.cert.org/stats/,January 2006.

[3]. Chris Anley. Advanced SQL Injection In SQL Server Applications. In An NGSSoftware Insight Security Research (NISR) Publication, 2002.

[4]. Cesar Cerrudo. Manipulating Microsoft SQL Server Using SQL Injection. Technical report, Application Security, Inc., 2002.

[5]. Martin Eizner. Direct sql command injection. Technical report, The Open Web Application Security Project, 2001. http://qb0x.net/papers/MalformedSQL/sqlinjection.html.

[6]. Mitchell Harper. Sql injection attacks - are you safe? Technical report, DevArticles, may 2002. http://www.devarticles.com/content.php?articleId=138&page=2.

[7]. Stuart McDonald. Sql injection: Modes of attack, defence, and why it matters. Technical report, The SANS Institute, jul 2002. http://www.sans.org/rr/appsec/SQL_injection.php.

[8]. Aaron C. Newman. Protecting oracle databases. Technical re-port, Application Security, Inc., 2001. http://www.appsecinc.com/presentations/Protecting_Oracle_Databa ses_White_Paper.pdf.49

[9]. William A. Qualls. Exploit in action: A beginners view of incident handling for sql injection techniques. Technical report, SANS Institute, 2003. http://www.giac.org/practical/GCIH/William_Qualls_GCIH.pdf.62

[10]. Kevin Spett. Security at the next level - are your web applications vulnerable? Technical report, SPI Dynamics, 2002. http://www.spidynamics.com/whitepapers/webappwhitepaper.pdf.

[11]. Halfond, W. G. J. and A. Orso (2005). AMNESIA: analysis and monitoring for Neutralizing SQL-injection attacks. . ASE'05. Long Beach, California, USA.

[12]. G.T. Buehrer, B. W. Weide. And P. A. G. Sivilotti (2005). Using parse tree validation to prevent SQL injection attacks. Proceedings of the 5th international workshop on Software engineering and Middleware. Lisbon, Portugal, ACM: pp. 106-113

[13]. Kemalis, K. and T. Tzouramanis (2008). SQL-IDS: a specification based approach for SQLinjection detection. SAC'08. Fortaleza, Ceará, Brazil, ACM: pp. 2153 2158.

[14]. MeiJunjin (2009). An approach for SQL injection vulnerability detection. Sixth International Conference on Information Technology: New Generations: pp. 1411-1414.

[15]. R. Ezumalai, G. A. (2009). Combinatorial Approach for Preventing SQL Injection Attacks. 2009 IEEE International Advance Computing Conference (IACC 2009). Patiala, India: pp. 1212-1217.
[16]. Hirschberg, D. S. (1975). "A linear space algorithm for computing maximal common sub sequences." A.C.M 18(06): pp. 341-343.
[17]. K. Amirtahmasebi, S. R. Jalalinia, S. Khadem, "A survey of SQLinjection defense mechanisms," Proc. Of ICITST 2009, vol., no., pp.1-8, 9-12 Nov. 2009.
[18]. Shubham Shrivastava, Rajeev Ranjan Kumar Tripathi, Attacks Due to SQL injection & their Prevention Method for Web-Application, International Journal of Computer Science and information Technologies, Vol 3 (2), pp.3615-3618, 2012.