Detection & prevention of SQL injection & Cross - site scripting attacks using SPWEPTLU technique

Sayed Yousuf Mahbobi, Amjad Khan, Mattiullah Nadiry,Ahmad Shekib Ghawsi

Abstract— SQL (Structured Query Language) injection is the most common and potentially hazardous attack that allows the attackers to fully manage the database by injecting or passing different malicious statements to the database engine in order to manipulate the data irresponsibly. This penetration to the system can cause serious damages such as stealing sensitive information, causing corruption in an organization or dismantling organization's operations. On the other hand XSS (Cross Site Scripting) is another type of security vulnerability that empowers the attackers to place client side scripts into a web pages visited by the users. In this paper we present optimal solution for detecting and preventing SQL and XSS injection attacks by restricting stored procedures with execute permission only to legitimate users.

The paper is organized as: Part-I is dedicated to the brief introduction of SQL & XSS attacks. In Part-II & Part-III a complete introduction of SQLIA along with different types of SQL attacks are explained. In Part-IV XSS is described briefly while, in Part-V relevant literature has been explored. In Part-VI the solution along with implementation are explained in the form of algorithms and flow charts whilst, in the last part conclusion and future work are illustrated.

Index Terms— SQL, Structured Query Language, XSS, Cross Site Scripting, SQLIA, Structured Query Language Attack.

1 Introduction

In Modern Times, World Wide Web has shaped the organization's infrastructure and becoming the essential part of each organization. Therefore, with the advancement and wide-spread usage of internet and World Wide Web, data-protection has become a challenging task. Since malicious users also known as cyber criminals are risen significantly and performing sophisticated logical attacks to harm the organizations by exploiting or auditing required information. The most popular and perilous attacks that are still lurking over the millions of websites or web applications are SQL & XSS attacks. In SQL attack, attackers usually distort the parameters of the SQL statements that are delivered to the backend database in order to perform the desired or intended malicious operations such as deleting, auditing and retrieving sensitive information through injections. XSS is another form of security vulnerability that targets the users by enabling them to view the pages that are affected awkwardly by client side scripts injections. XSS attacks are utilized for stealing cookies and hijacking sessions in a web application that grant the attackers to bypass the application rules and take the full control of it. As a result, SQL injection and XSS attacks target all the applications connected to the databases and compromise the data security hence, it can completely damage the organizations assets and resources.

- *Sayed Yousuf Mahbobi graduated from bachelor of computer science department of Kardan Univeristy, Afghanistan, PH-0093704076468. E-mail: y.mahbobi@outlook.com*
- *Amjad Khan is PHD scholar at Abdul Wali Khan University, Pakistan PH-00923219033960. E-mail: amjad@aup.edu.pk*
- *Mattiullah Nadiry graduated from bachelor of computer science department of Kardan Univeristy, Afghanistan, PH-0093766040566. E-mail: mattiullah.nadiry@gmail.com*

2 Defination OF SQLI

SQL Injection is the interpolation of the SQL statements injected to the application's input boxes which causes severe harms to the data stored in the database and a major threat to the application security. Lack of supplying sufficient security in applications and lack of sufficient programming knowledge causes attackers to be successful in penetrating to the application and performing SQL attacks. In SQL Injection attack, the attackers append a harmful string input through the application's entry point or input boxes, which transforms or manipulate the original SQL statement to the SQL statement exploitable by the attackers. SQL injection can sabotage the database in peculiar manners such as unauthorized data manipulation or even in most severe cases execution of system level commands that causes denial of services to the application thus, it loses the system confidentiality and trustworthiness.

3 Types of SQLI

3.1 Tautologies

In tautology-based attacks, attackers append one or more conditional SQL statements into the query in order to make SQL command assess to true condition such as (2=2) or (''=''). The most frequent usage of this approach is to bypass authentication on web pages concluding in access to the database. The SQL query demonstrated in Fig. 1 shows the tautology SQLIA.

```
SELECT employee_password
FROM tblEmployee WHERE
Employee_ID = '2' OR
'2'='2' -- ' AND
employee_password
='he@sa';
```

Fig1: Tautology attack

3.2 Piggy-backend query

In Piggy-backend query, attackers terminate the original query statements by using query delimiter, such as ";" and inject additional query statements to the end of original query. In this technique the first query is original whereas the subsequent queries are injected. Piggy backend query attacks are very disastrous because attackers can append any sort of harmful statements. The SQL query in the below figure demonstrate the piggy backend query attack.

```
SELECT Employee_ID FROM
tblEmployee
WHERE Employee_ID = 8
AND Employee_Password =
'abc';
DROP TABLE tblEmployee
```

Fig2: Piggy-backend attack

3.3 Logically incorrect

Logically Incorrect attack arise by exploitation of handy information like error messages that comes from the database for unauthentic query. This erroneous information helps the attackers to find hole in the system and perform attack. SQL query mentioned in the below figure describes Logically Incorrect attack.

```
SELECT * FROM
tblEmployee
WHERE Employee_ID =
'111' AND Employ-
ee_Password = 'abc'
AND CONVERT (int, ‘a’)
```

Fig3: Logically incorrect attack

3.4 Union query

Union query injection or in other word statement injection attack. In which, attackers embed supplementary statements into the original SQL query. Union query attack occurs by embedding UNION keyword into unprotected parameter, as shown in Fig 4. Resulting the database to return dataset which is the union of result of original query with the result malicious query.

```
SELECT Student ID FROM
tblStudent UNION SELECT
Teacher ID FROM tbl-
Teacher
```

Fig4: Union query attack

3.5 Stored procedure

In stored procedure attack, attackers concentrate on the stored procedures which exists in the database system. Stored procedures are near to the database engine thus it runs directly by the database engine. It is an exploitable piece of code that returns either true or false for the authorized or unauthorized users. For SQLIA, attackers call the stored procedure and append the command for instance the “; SHUTDOWN; --" command to stop the database from functioning. The SQLI query in the below figure, shows the stored procedure attack.

```
EXEC PROC Test; SHUTDOWN;
```

Fig5: Stored procedure attack

3.6 Inference

Inference attack allows the attackers to turn the nature of a database or application. There are basically two types of inference attack.

3.6.1 Blind injection

SQLIA happens when programmers fail to remember to conceal an error messages which cause data insecurity, this error message aids SQLIA to harm the database by querying series of logical inquiry through SQL statements. The below figure demonstrates the blind injection attack.

```
SELECT password FROM
empTable WHERE username
= 'user1' AND 2=1 --
AND password = AND pin =
2 SELECT info FROM empT-
able WHERE username
='user1' AND = 2 -- AND
password =3
```

Fig6: Blind injection attack

3.6.2 Timing attack

Timing attack enables attackers to accumulate information from a database by observing timing procrastination in the database's responses. This type of attack utilize if condition statement to gain a time procrastination purpose such as WAITFOR, which causes the database to postpone its response by a specified time. The below figure displays timing attack.

```
DECLARE @name VARCHAR
(500);
SELECT @name = empName
FROM tblemployee
IF(substring(@name,0,2))
> 0
Waitfor delay '0:0:8'
```

Fig7: Timing injection attack

3.7 Alternate encoding

In alternate encoding, attackers customize the injection query using alternate encoding, such as hexadecimal, ASCII, and Unicode. This approach permits attackers to escape from developer's filter and perform any sort of SQLIA. When this type of attack couples with other attack techniques it could be powerful, because it can target various layers in the application so developers need to be familiar to all of them to supply an effective defensive measure to avoid the alternate encoding attacks. The SQL query, which is shown in Fig 8, describes the alternate encoding attack.

```
SELECT * FROM Accounts
WHERE acc_id ="AND
pin=1; exec
(char(0x23571324f2134124
wwa33))
```

Fig8: Alternate encoding attack

4 Defination of XSS

Cross Site Scripting is a client-site script injection attack which attempts to run malicious code in the browser of the victim by passing legal code in a web page or web application that can be run later when the user actually lands on the web page or web application infected by the malicious code. The web page or web application acts like a vehicle to carry the malicious script to the user's browser. These vehicles can be forums, message boards, and web pages that allow comments.

XSS attacks are most common in JavaScript, primarily because JavaScript is recognized language for most browsers.

What can the attackers do with JavaScript?

1. Malicious JavaScript has access to all the contents of the web page like access to the user's cookies. Cookies are often utilized to save session tokens. If the attacker succeed to attain the user's session cookie then they can impersonate that user, perform actions on behalf of the user, and gain access to the user's sensitive data.
2. JavaScript can read the browser's DOM and make the desired customizations to it.
3. JavaScript utilizes the XMLHttpRequest object to forward HTTP requests with arbitrary content to arbitrary destinations.
4. JavaScript in modern browsers utilizes HTML5 APIs. For instance, it can access the user's webcam, location, microphone, and even particular files from the user's file system.

5 Literature Review

AMNESIA (Analysis and Monitoring for Neutralizing SQLIA) technique was developed by W. G. J. Halfond et al [1] that detects and prevents SQLIA at runtime based on two analysis phases namely dynamic and static. In the static analysis it generates types of query statements as a model and in dynamic analysis phase it interprets all the queries against static model before they are sent to the database.

SQL syntax-aware at web application layer to evaluate query strings in web application server and negative impact at the database layer to catch the untrusted data technique was developed by A. Alazab et al [2] but proved having network overhead.

SAFELI is a tool presented by X.Fu et al [3], which identified the SQL Injection attacks at compile time from the source code. The drawback of this tool was it could not prevent tautologies attack.

WASP (Web Application SQL Injection Protector) tool was developed by W. G. J. Halfond et al [4] which was potent in ceasing more than 12,000 attacks without generating any issues in database layer. The limitation of this tool can be founded by deploying web applications.

R-WASP (Real Time-Web Application SQL Injection Detector and Preventer) tool was developed by M. H. A. S. P. Medhane [5], which could cease all attacks potentially and detects SQLIAs in real-time environment. The limitation of this tool is required more practice to work efficiently.

Suitable Real Time Web Application SQL Injection Protector (RT-WASP) tool was developed by N.S. Ali et al [6] to detect SQL injection attacks in stored procedures. The drawbacks of RT-WASP tool was that it did not detect the XSS attacks.

Principle of dynamic query structure validation which was done through analyzing query's semantics was proposed by S. Manmadhan et al [7]. The main aim of this technique spotlighted on the particular type of attacks which was proved vain later.

SecuriFly is a prevention tool for java developed by M. Martin et al [8]. This tool is utilized to check string for malicious information and tried to fix it before passing to the database. The limitation of this tool was complication in finding all the sources of user input.

JDBC-Checker proposed by C. Gould et al [9], which is technique for statically analyzing the validity of dynamically generated SQL queries. The drawbacks of this technique was that it could not identify general types of SQLIA because the attackers usually writes queries accurately.

Dynamic Candidate Evaluations method was presented by P. Bisht et al [10]. This technique separated out the query structures from each SQL query locations at the run-time. The drawback of this technique was partially cease SQLIAs due to the constraints on the fundamental method.

Swaddler method that analyzed the inner state of a web application was proposed by Macro Cova et al [11]. Firstly, the method interpreted the normal values for the application's state variables. At the detection phase, the method control the application execution to find abnormal states. The disadvantage of this method was that it partially identified SQLIAs.

DIWeDa, which was a mock-up that acted at the session level to find malicious input in Web applications presented by A. Roichman et al [12]. The drawback of this technique was that it couldn't detect all types of SQLIA.

Positive Tainting and Syntax Aware Evaluation technique was proposed by William G. Halfond et al [13]. In this technique it basically differentiated the strings generated by the java and the strings originated from external sources to identify the trusted and untrusted data. If untrusted found then by syntax-aware it prevented the data from being passed to the server. The main drawback of this approach was initialization of trusted strings by developers.

SQL Prevent, which was consist of a HTTP request interceptor proposed by P.Grazie [14] The HTTP requests were stored into the local storage. Then, SQL interceptor intervened the SQL statements and moved them to the SQLIA detector. The main problem of this approach was extra storage and processing that effect performance state.

Automated approaches that were based on defensive programming in which the inputs were filtered to avoid user from inserting harmful keywords or characters proposed by Mei Junjin [15], but proved unable to detect the stored procedure and alternate encoding attacks.

SQLIPA that exploits hash value method to improve user authentication mechanism was presented by S. Ali et al [16]. The drawback of this technique was detecting merely Tautologies attacks.

Usage of prepared Statement was proposed by Stephen Thomas et al [17]. Utilizing JDBC for database connectivity, the Prepared Statement on the spot liberate the special characters before implementing the query. The main drawback of this approach was that it could not stop all types of SQLIA.

PDO (PHP Data Object) that defined a lightweight, persistent interface for integrating with databases in PHP was suggested by M. Sendiang et al [18], Parameterized Queries in PDO could merely prohibit tautologies and union attacks.

Data validation and database lockdown proposed by M. Zabi et al [19] in order to minimize the SQL injection in Microsoft Internet information server. The main drawback of this approach was that it could only stop tautologies attacks.

Static and dynamic analysis was proposed by Lee, Inyong et al [20] in which SQL query is cleansed during the runtime and then the query was matched with the predefined SQL query. This approach can abstain all types of SQL attacks except real-time SQL attacks.

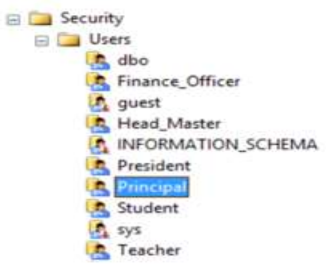
New methodology was proposed by Sailor Pratik et al [21] In this approach the application was blocking the common keywords such as union, special characters, delimiters and so on with the impression that it will be comparatively better and easy approach but it raised unauthentic alarm.

Runtime controlling approach progressed by Ramya Dharam [22] was presented and evaluated to detect and prevent tautologies attack in web applications. Their view was to not only validate the client side code but also we have to validate the server side code during runtime.

6 PROPOSED SOLUTION & IMPLEMENTATION

The SPWEPTLU (Stored Procedure with execution permission to legitimate users) technique helps in detecting and preventing SQLI, XSS (Cross-Site Scripting) and other types of attacks that can be performed via application penetration. This approach consists of the following steps.

1. Identify the organization business entities roles and create separate user login for each roles.



IJSER

Fig9:Logins

In figure 9, we identified the business entities roles such student, teacher, president etc. And then created logins for each mentioned entities under the security tab.

2. Write stored procedure.

```
CREATE PROC
sp_CheckStudentAppraisl_Record_
Exists
@Teacher_ID VARCHAR(50),
@Class_ID numeric(18,0),
@Section_ID numeric(18,0),
@Subject_ID numeric(18,0),
@Month_ID numeric(18,0),
@check_Record_Exists AS NUMER-
IC(18,0) out
AS
BEGIN
       SET NOCOUNT ON;
       -- Get the Current Ses-
       sion
       DECLARE @Session_ID AS
       NUMERIC(18,0);

       SELECT @Session_ID =
       Session_ID FROM tblSes-
       sion WHERE Year =
       YEAR(GETDATE())
       -- Get the Apprasial Re-
       sult If Exists
       DECLARE
       @Exists_StudentApprisal
       AS NUMERIC(18,0);
       -- Select the data from
       required tables
       SE-
       LECT@Exists_StudentAppri
       sal =
       COUNT(Apprisal)FROM
       tblStudent_Performance
       WHERE Enroll_ID IN (SE-
       LECT
       tblEnrollment.Enroll_ID
       FROM tblEnrollment INNER
       JOIN tblOfferedSubjects
       ON
       tblEnroll-
       ment.Class_Subject_ID =
       tblOfferedSub-
       jects.Class_Subject_ID
       INNER JOIN tblClass ON
       tblOfferedSub-
       jects.Class_ID =
       tblClass.Class_ID INNER
       JOIN tblStudent ON
       tblEnrollment.Student_ID
```

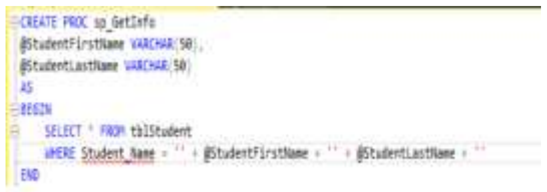
```
         = tblStudent.Student_ID
   WHERE(tblEnrollment.Session_ID
          = @Session_ID) AND
  (tblClass.Class_ID = @Class_ID)
  AND (tblEnrollment.Section_ID =
          @Section_ID) AND
  (tblOfferedSubjects.Subject_ID
  = @Subject_ID) AND
   (tblEnrollment.Teacher_ID =
   @Teacher_ID) AND (tblStu-
   dent.Student_Current_Status =
   1)) AND Month_ID = @Month_ID
   IF (@Exists_StudentApprisal =
  0)
    BEGIN
     SET @check_Record_Exists = 0
    END
   ELSE IF
  (@Exists_StudentApprisal !=0)
    BEGIN
     SET @check_Record_Exists = 1
    END
  END
```

Fig 10: Database Procedure.

In figure 10 we write our procedure inside the database with the below mentioned rules in order to lock security flaws and prevent security attacks.

Rules of persistent and secure procedure writing in SQL

I. Do not use concatenation even inside the stored procedure with parameters because it is also vulnerable to security attacks though, sometimes the circumstances comes where we need concatenation of few columns, in that case we can use the concat() function which is secure way of concatenating the strings or columns. Alternative way can be using local variables inside the procedure.



```
CREATE PROC sp_GetInfo
@StudentFirstName VARCHAR(50),
@StudentLastName VARCHAR(50)
AS
BEGIN
    SELECT * FROM tblStudent
    WHERE Student_Name = '' + @StudentFirstName + '' + @StudentLastName + ''
END
```

Fig 11: Avoid concatenation.

II. Follow suggestions stated by Microsoft in the best practices article [23]. These suggestions may improve procedure performance and security.

3. Assign procedure to proper role.

```
GRANT EXEC ON
sp_CheckStudentAppraisl_Record_
Exists To Teacher
```

Fig 12: Grant procedure to role.

In figure 12 we grant execute only to those entities in an organization that mentioned procedure is part of their tasks or roles for example the teacher at the school can only be able to check students appraisal no other entities should be able to check or access the students appraisal by performing injection or any other sort of attacks so we grant the execute permission to teacher only for checking students appraisal. In this case if malicious user tries to inject any malicious statement, the database engine will stop him/her. For example the teacher bypasses the rules or penetrates into the system though the application and injects some series of harmful statements or queries, so the database will automatically not executing that statements and generates error.

```
EXEC
sp_CheckStudentAppraisl_Record_
Exists;
Shutdown;
```

Fig 13: User perform procedure call that is not part of his/her role + attack.

```
Msg 229, Level 14, State 5,
Procedure sp_AddYear, Line 1
[Batch Start Line 0].
The EXECUTE permission was de-
nied on the object
'sp_AddYear', database 'Khalid
Bin Walid', schema 'dbo'.
```

Fig 14: Denial message from database.

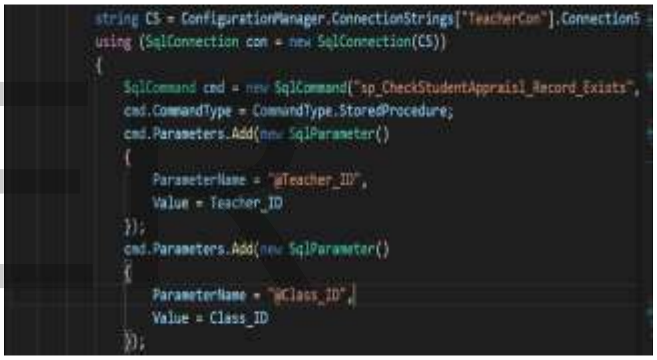
```
DECLARE @result AS NUMER-
IC(18,0);
EX-
ECsp_CheckStudentAppraisl_Recor
d_Exists 'KBW-
2020010005',2,1,1,1,@result
out;print @result; DELETE FROM
tblStudent_Performance;
```

Fig 15: User perform procedure call that is part of his/her role + attack.

```
Msg 229, Level 14, State 5,
Line 3
The DELETE permission was de-
nied on the object 'tblStu-
dent_Performance', database
'Khalid Bin Walid', schema
'dbo'.
```

Fig 16: Procedure executed but the attack denied by the database engine.

4. Finally, Calling procedure through application.



```
string CS = ConfigurationManager.ConnectionStrings["TeacherCon"].ConnectionS
using (SqlConnection con = new SqlConnection(CS))
{
    SqlCommand cmd = new SqlCommand("sp_CheckStudentAppraisl_Record_Exists",
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.Parameters.Add(new SqlParameter()
    {
        ParameterName = "@Teacher_ID",
        Value = Teacher_ID
    });
    cmd.Parameters.Add(new SqlParameter()
    {
        ParameterName = "@Class_ID",
        Value = Class_ID
    });
```

Fig 17: Calling procedure from app.

In figure 17, we include the proper connection according to the currently login user's role then we simply call the procedure with passing the arguments as a SQL parameters from application and binding it with the parameters that we predefined in SQL procedure in the database in order to treat every single parameter as a variable not as a series of string as we do in a normal query with concatenation or passing simple parameters to a query.

SPWEPTLU technique flow chart for preventing SQLIA

International Journal of Scientific & Engineering Research, Volume 12, Issue 1, January-2021
ISSN 2229-5518

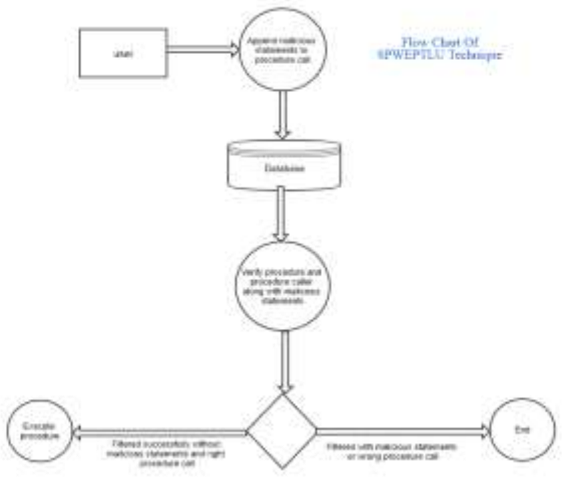

user
Append malicious statements to procedure call
Flow Chart Of SPWEPTLU Technique
Database
Verify procedure and procedure caller along with malicious statements
Execute procedure
Filtered successfully without malicious statements and right procedure call
Filtered with malicious statements or wrong procedure call
Exit

Fig 18: SPWEPTLU technique flow chart for preventing SQLIA.

Cross site scripting attack prevention

The SPWEPTLU (Stored procedure with execution permission to legitimate users) technique also helps to prevent cross site-scripting attacks because it stores the user input as a variable before passing it to the database and then it brings it back from the database in a string form hence, it will be displayed later in the DOM (document object model) as a string not as a script to be implemented.

```
<SCRIPT>
Var adr = ‘evil.php?monster=’+
escape(document.cookie);
</SCRIPT>
```

Fig 19: XSS attack.

So this information will be stored as a text using stored procedure and will be fetch back from database to the user interface as text and will have no effect on script of the page thus, it prevents from XSS attacks.

SPWEPTLU technique flow chart for preventing XSS

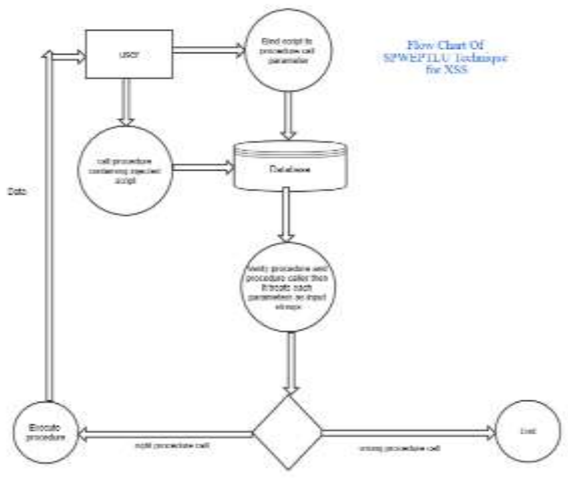

user
Bind script to procedure call parameter
Flow Chart Of SPWEPTLU Technique for XSS
call procedure containing injected script
Database
Data
Verify procedure and procedure caller then filtrate each parameter as input strings
Execute procedure
right procedure call
wrong procedure call
Exit

Fig 20: SPWEPTLU technique flow chart for preventing XSS.

Advantages of Proposed Solution

i). Security reinforcement: Restricting permissions in the database helps security reinforcement.

ii). Concealed logic: In the proposed solution all the logics implemented internally in the database are hidden from the users.

iii). Denial of Arbitrary Actions: The users are not able to perform any action arbitrarily because they don’t have any information related to any table or any column, in fact the users are just blindly executing assigned procedures that they are not informed of what is written inside of it.

iv). Permission restriction: The users are not able to perform anything because they only have execution permission to assigned procedures but they don’t have the permission to view the procedure definition or to view any column in fact the users don’t know whether there exists any table, procedure, function or view in the database or not.

v). Performance optimization: Stored procedures are executed quickly because, they are near to the database engine therefore we save time to access database certain times for different operations as we do in a normal query.

vi). Automatic denial of attacks: In the proposed solution, the database itself cease the attacks when it encounters.

vii). Data accuracy and consistency: data can be manipulated by accurate user and consistency exists in the data because it cannot be changed by any attacks.

viii). XSS attacks prevention: the proposed solution prevents from cross site scripting attacks because the JavaScript that the user passes to the server will be stored as a parameter string thus, it cannot be optimized by the attackers.

ix). Data security: As data are stored in such a way that users are not aware of its existence therefore the users would not be able to do anything. This approach firms data security.
Implementable by all relational databases such as Oracle, MYSQL, SQLSERVER and so on.

7 Conclusion & Future work

There are many vulnerable applications in today's world, because the programmers mainly focus on user interfaces and functionalities to produce a software irrespective of focusing on security which is the core part of software development and one of such security threat is SQL injection and XSS attacks through which attackers penetrate into the system and harm the organization. Thus some organizations use various inaccurate techniques as discussed trivial by the researchers we mentioned above such as parameterized query, wrongly usage of stored procedure, using available software packages in the market that claims to prevent SQLIA. Hence the paper introduce optimal approach for not only detecting the SQL injection and XSS attacks but also to prevent it from happening by using the stored procedure with execute permission to legitimate users technique that helps to prevent any sort of application wise security attacks.

The future work for this paper is to enhance the capability of this approach to detect and prevent SQLIA at application level.

References

[1] W. G. J. Halfond and A. Orso, "Preventing SQL Injection Attacks Using AMNESIA," Presented at the Proceedings of the 28th International Conference on Software Engineering (ICSE), ACM, Shanghai, China.

[2] A. Alazab , A. Khresiat , " New Strategy for Mitigating of SQL Injection Attack", International Journal of Computer Applications (IJCA), Volume 154, paper No.11.

[3] X.Fu, X. Lu, B. Peltsverger, S. Chen, G. Southwestern, K. Qian, and S. Polytechnic, "A Static Analysis Framework for Detecting SQL Injection Vulnerabilities" 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), IEEE, ISSN: 0730-3157, pages Number 1–8. , China.

[4] W. G. J. Halfond, A. Orso, and I. C. Society, "WASP: Protecting Web Applications Using Positive Tainting and SyntaxAware Evaluation", IEEE Transactions on Software Engineering, volume. 34, Issue 1, pages. 65–81, 2008.

[5] M. H. A. S. P. Medhane, "R-WASP: Real Time-Web Application SQL Injection Detector and Preventer", International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-2, Issue-5, pages. 327– 330.

[6] N.S. Ali, A. Shibghatullah, "Protection Web Applications Using Real-Time Technique to Detect Structured Query Language Injection Attacks", International Journal of Computer Applications (IJCA), Volume 149, paperNo:6.

[7] S. Manmadhan , Manesh T. , "A Method of detecting SQL Injection Attack to Secure Web Applications", International Journal of Distributed and Parallel Systems (IJDPS) ,Volume.3, Issue.6.

[8] M. Martin, B. Livshits, and M. S. Lam., "Finding Application Errors and Security Flaws Using PQL: A Program Query Language" ACM SIGPLAN Notices, Volume: 40, Issue: 10 Pages: 365-383.

[9] C. Gould, Z. Su, and P. Devanbu. JDBC Checker, "A Static Analysis Tool for SQL/JDBC Applications", in Proceedings of the 26th International Conference on Software Engineering (ICSE04) Formal Demos, ACM, ISBN: 0-7695-2163-0, pages 697– 698.

[10] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan, "CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks", ACM Transaction on Information System Security, pages.1–39.

[11] Macro Cova, Davide Balzarotti." Swaddler: An Approach for the Anomaly-based Detection of State Violations in Web Applications", In Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID), pages: 63–86.

[12] A. Roichman, E. Gudes, "DIWeDa - Detecting Intrusions in Web Databases". In Proceeding of the 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Springer, volume. 5094, pages. 313–329, Heidelberg.

[13] William G. Halfond, Alessandro Orso, "Using Positive Tainting and Syntax Aware Evaluation to Counter SQL Injection Attacks", 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, ACM. Pages: 175 – 185.

[14] P.Grazie., PhD, "SQL Prevent Thesis", University of British Columbia (UBC) Vancouver, Canada.

[15] Mei Junjin, "An Approach for SQL Injection Vulnerability Detection" Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations, IEEE Computer Society, Las Vegas, Pages 1411-1414.

[16] S. Ali, SK. Shahzad and H. Javed, "SQLIPA: An Authentication Mechanism against SQL Injection", European Journal of Scientific Research, Volume.38, Number.4, pages: 604-611.

[17] Stephen Thomas, Laurie Williams, "Using Automated Fix Generation to Secure SQL Statements", Proceedings of the Third International Workshop on Software Engineering for Secure Systems (SESS '07), page 9.

[18] M. Sendiang, A. Polii, J. Mappadang, "Minimization of SQL Injection in Scheduling Application Development", International Conference on Knowledge Creation and Intelligent Computing (KCIC), IEEE, Indonesia.

[19] M. Zabi, M.Joseph "Minimization of SQL Injection", International Conference of Technology Sight, IEEE, Saudi.

[20] Lee, Inyong, Soonki Jeong, Sangsoo Yeo, and Jongsub Moon. "A Novel Method for SQL Injection Attack Detection based on Removing SQL Query Attribute Values." Mathematical and Computer Modelling, Volume 55, Issues 1–2, Pages 58–68.

[21] Pratik H Sailor, Prof. Jaydeep Gheewala. "Detection and Prevention of SQL Injection Attacks", International Journal of Engineering Development and Research (IJEDR), ISSN: 2321-9939, Vol.2, Available: http://www.ijedr.org/papers/IJEDR1402215.pdf.

[22] "Runtime Monitors for Tautology based SQL Injection Attacks", Ramya Dharam, Sajjan G.Shiva, "International Journal of Cyber Security and Digital Forensics (IJCSDF).

[23] "Microsoft Suggestion for Proper Procedure Writing", Available: https://docs.microsoft.com/en-us/sql/t-sql/statements/create-procedure-transact-sql?view=sql-server-ver15#see-also