A Method to Prevent SQL Injection Attack using an Improved Parameterized Stored Procedure

Kamsuriah Ahmad[1], Mayshara Karim[2]
Center for Software Technology and Management
Faculty of Information Science and Technology
Universiti Kebangsaan Malaysia, 43600, Bangi, Selangor, Malaysia

***Abstract*—Structured Query Language (SQL) injection is one of the critical threats to database security. The effects of SQL injection attacks cause the data contained in the database to be at risk of being exploited by irresponsible parties, compromising data integrity, disrupting server operations and in return affecting the organization's image. Although SQL injection is an attack performed at the application level, SQL injection prevention requires security controls at all levels, namely application level, database level and network level. The absence of SQL injection prevention measures at the application level makes the database vulnerable to attack. Reviews indicate that the current approaches still not sufficient in addressing these three issues, which are i) improper use of dynamic SQL, ii) lack of input validation process and iii) inconsistent error handling. Currently, program and database code security is based solely on basic security measures that are focused at the network level such as network firewalls, database access control and web server request filtering. Unfortunately, these measures are still inadequate and not sufficient to safe guard the program code and databases from the attack. To overcome this shortcoming as addressed by these three issues, a new comprehensive method is proposed using an improved parameterized stored procedure to enhance database security. Experimental results prove that the proposed method is able to prevent SQL injection from occurring and able to shorten the processing time when compared with existing methods, hence able to improve database security.**

Keywords*—SQL injection prevention; database security; parameterized stored procedure; network firewall

I. Introduction

The sophistication of information technology makes life easier but at the same time poses a threat if the security aspect is not given special attention [1]. The internet that connects people around the world carries the threat of hackers. SQL injection is used as a method to steal and exploit information on the database [2]. SQL injection can affect organizational data security, and now they start to realize the importance of preventing this attack before it happens [3]. The prevention from this injection has become the focus of database administrators, network administrators, application code programmers, database system providers, software developers and the top management of the organization. The problems brought by SQL injection have been around for a long time and are still a hot topic in information security issues [4] [5]. Web application security and database security are inter-dependent. If a web application is manipulated by hackers, databases that have been equipped with security features can still be exploited. There are a number of threats to database security: among them are i) excessive privilege which refers to users who are given permission to access or carry out various transactions in the database but abuse the permission instead, and ii) SQL injection which is the entry of unauthorized input into the database to carry out any instructions that are not valid [6]. In addition, weak audit trails or automatic recording of database transactions that are not performed properly and media used as storage for backups such as hard disks and tapes are also prone to theft [7]. Therefore to improve database security, this paper aims to propose a new method in preventing SQL injection. To discuss the existing prevention methods on SQL injection and the motivation of this work, this paper is organized as follows: section II discusses the issues of SQL injection attack. Section III describes various ways that cause web applications to be attacked by SQL injection. Section IV highlights the impact of SQL injection on database security. Section V reviews the existing SQL injection prevention method. Based on the limitations faced by the existing methods, a new method is proposed in Section VI. Section VII explains the experiment conducted to prove the efficiency of the proposed method and Section VIII concludes the study and future work suggestion.

II. SQL Injection Attack

SQL is a standard programming language used to access databases. SQL injection is the act of putting malicious code as an input to a web application and sending it to a database to execute various commands [8]. An attack occurs when a hacker exploits an SQL injection to execute an unauthorized command [9]. Examples of common exploits are: stealing confidential information that can bring profits such as credit card numbers, bank savings account numbers, passwords, business transaction records and medical records. In addition, exploitation can also involve data modification [10]. For example; students trying to change grades or exam marks. There are also cases which are not personal attacks; such as sabotage the database by deliberately deleting certain tables, shutting down database operations and disrupting network traffic [4]. The term ‘injection’ is used because malicious code which considered as a non-valid input is injected into a valid SQL statement. The database will execute this command because it is valid in terms of the syntax and rules [11]. Fig.1 explains the scenario of SQL injection attack.

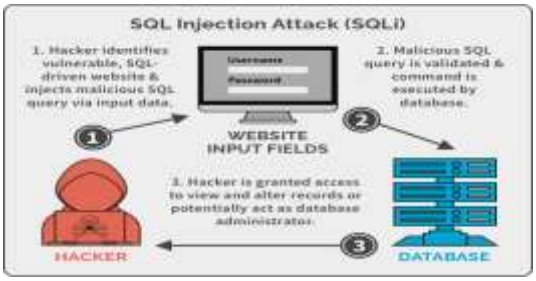

Fig. 1. SQL Injection Attack [2].

The term attack in the context of SQL injection is defined as an unauthorized access to applications or systems which normally done by hackers [12]. This access is obtained through SQL injection mechanisms derived from SQL query modifications [9]. Web applications that have user input can be attacked by SQL injection because the application code has certain vulnerabilities that expose to the attacks.

This will lead to the occurrence of bugs, loopholes, vulnerabilities or defects. These weaknesses shall be manipulated by unauthorized users to gain unrestricted access to stored data [10]. Therefore, a proper mechanism is needed to prevent the attack, such as validate the user input both at the client and server side [13] [14]. Since lack of proper mechanism in preventing the attack at the application and database level exists in the literature, therefore it has become the motivation of this study. In order to understand in details, next section will discuss the types of SQL injection that usually attack databases.

III. SQL Injection Mechanism

Web applications have many loopholes and make them vulnerable to SQL injection attacks. There are various ways that cause web applications to be attacked by SQL injection. Among them are discussed as follows.

- SQL Injection through error messages.

An error-based injection is a type of SQL injection derived from an error message to figure out the structure and the type of database [15]. The attacker intentionally enters wrong input logically in order to allow database generates an error message. This error message contains information that allows the attacker to identify the parameters vulnerable to the injection.

- Boolean-based blind SQL injection

The term blind means that the SQL injection is performed when the programmer has set a generic custom error message in case web application encounters an error [16]. Without displaying error messages, database vulnerabilities can be protected. The hacker has to deal with a database system that does not display error messages and as an alternative, hackers submit a series of "TRUE" and "FALSE" queries via SQL queries [17]. Information about the database will be revealed through the results of these queries.

- Time-based blind SQL injection

Time-based SQL injection means hackers obtain information on database based on response times. SQL command is sent to database with code to force the database to wait for a specified amount of time during the execution of the queries. The response time indicates whether the result of the query is true or false. While waiting for the query to be processed, the attacker able to execute another query and might inject malicious code in the query [18].

- UNION-based SQL injection

The hacker connects the SQL injection to the original SQL query by using the UNION clause to retrieve data from another table. The result of this injection is that the database provides a data set that contains a combination of the results of the original query and the results of the SQL injection query [19].

To avoid SQL injection through error messages will be the focus of this study. Hackers have a variety of reasons and motivations when it comes to hacking. Hackers gain access and control to databases illegally through SQL injection. With such access hackers can perform various actions to manipulate the data stored in the database. The main impact when SQL injection occurs is the disruption of the confidentiality and integrity of the data in the database [20]. Table I describes the actions that hackers can take and the consequences of the actions taken.

TABLE I. The Action and the Effects of SQL Injection Attacks

	Action	Effects
1	Identify parameters that can be injected	Attackers able to find input parameters that are vulnerable to SQL injection
2	Perform a fingerprint on the database	An attacker can find out the type and version of the database being used. Easier to devise a more specific attack strategy.
3	Identify database schema	Attackers able to extract the information on database schema accurately
4	Extraction of data	Attackers able to extract complete data from database table
5	Add, remove and edit data	Attackers can alter certain data for their own benefit

IV. The Impact of SQL Injection Attack on Database Security

Most organizations are unaware that their web applications are vulnerable to SQL injection attack or have been attacked by SQL injection [21]. Awareness and knowledge of SQL injection is still lack in the organization, any prevention steps are mostly at the basic level such as network firewall installation, the use of Secure Socket Layer (SSL) and network access control [2]. However, these measures only provide protection at the network level, and not at the application level [11]. Reviews state that the weakness in preventing SQL injection lies in the programming code of the web application itself [15]. Although various types of techniques have been introduced, most of the researchers insist that the application program code needs to be ensured its security first [22]. Firewalls and protocol information unable to protect web applications from hackers due to their position are behind the web application infrastructure [12]. The risks of attacks are increased due to the exposure of web application, firewall-friendly HTTP protocols and lack of database security [23].

Even though, there are approaches to improve database security through SQL injection prevention both at application and database level, however they are not capable to protect database against these three attacks which are:

1) vulnerabilities triggered when using dynamic SQL[13][18].
2) malicious code entered through input data [5][6].
3) inappropriate error handling message [10][14].

The cause of these attacks is explained as follows:

- Dynamic SQL usage

Dynamic SQL is a query language to build SQL statements and performs functional logic such as data search, user login and others, where this query is created dynamically at run-time. Although, these language are beneficial and user-friendly; but at the same time they expose the web applications to SQL injection. Often web applications are vulnerable to the attacks when dynamic SQL statements are being used [7]. Because, this language is dynamic in nature, therefore, an unauthorized user can modify the original query by injecting a malicious code to the query at run-time [24]. The following shows the impact of using dynamic SQL on database security. An example is used to show how SQL injection attacks might occur when using dynamic SQL. During runtime, the following dynamic SQL statement is sent by the web server to the database server:

SELECT * FROM program WHERE ID = 9

The intention of this query is to list the data from all the columns in a table named program; where column ID is 9. Since the query is dynamic, an hacker with a good knowledge of programming able to manipulate the query by injecting malicious code and modify the query dynamically at runtime, for example adding the symbol (') after the number 9. SQL statement is changed to

SELECT * FROM program WHERE id = 9'

This example shows that the use of dynamic SQL gives an opportunity to the unauthorized user to perform attack on database. Therefore the used of dynamic SQL should be avoided [18]. Since most organization used stored procedure with dynamic SQL [10], therefore, a preventive step at the database level is needed to avoid SQL injection attempts.

- Input validation

Web applications allow users to log in into the system. However, the data entered by the user may contain some malicious code and can easily exploit through SQL injection [25]. The inputs need to be checked and validated beforehand. The absence of an input validation process will contribute to the vulnerability of SQL injection. When a web application ensures that the received input is within the expected range, the malicious code could be prevented from entering the database server thereby preventing SQL injection. Therefore, a proper mechanism is needed to validate the input in order to prevent malicious code from entering the database server.

- Error handling process

During data processing, system will prompt an error message if they encounter problems. However, the default error messages reveal sensitive information and weaknesses of the database. The intruder of the system will learn from these errors, and this will give an opportunity for them to breach the system [14].

The limitations of the existing prevention methods at the application and database levels in the organization have become the motivation of the study. An improved method that able to overcome these three issues discussed above is needed in enhancing organization security level.

V. An Overview of the Existing SQL Injection Prevention Method

Measures to improve the security of web applications and databases can be done at the server, network, database and application levels. However, most organizations take the basic security measures at the server and network level only [26]. The server and ICT network level is the physical level which is also an asset subject to the security procedures set by the existing policies. For example, organizations in the public sector need to comply with the Security Policy which outlined rules for internet access control, use of firewalls, server configuration settings, user access control and others.

Since this study focuses on the improvement of SQL injection prevention at the application and database level, the reviews of the existing methods are based on these two levels, which are summarized in Table II. Existing methods are reviewed based on their ability to address the threats causes by the three issues discussed namely i) dynamic SQL usage, ii) data validation at both client and server side, iii) error handling process. These three issues need to properly address to prevent SQL injection.

As Table II indicates, the existing methods did not address the three issues highlighted in one single method; therefore they are not sufficient in securing the database from SQL injection attacks. The method proposed by [4][12][21][24] validates the data at the client side and ignore the validation process at the server side. Therefore malicious code might enter the database. Even though methods proposed by [6][23][27][28] used parameterized query however dynamic SQL is still used. Therefore, this method is unable to prevent unauthorized user from modifying the original query by injecting malicious code. The used of dynamic SQL should be avoided to increase database security [2]. Dynamic analysis as used in [25] increases the query processing response time. As an organization that constantly deals with users through web applications, fast response times are essential. There are previous researchers who embedded handled error messages in their approaches, but the usage is inconsistent since the data is not validated at the application and database level [2][14]. Therefore these approaches are not sufficient in dealing with SQL injection attacks.

TABLE II. EXISTING APPROACHES IN SQL INJECTION

Author	Method used	Remarks
[2]	Error handling	Users are able to access database information through error messages. However, the error handling messages are not done consistently
[4]	Input Validation	The validation is only at the application side
[6]	Parameterized query	The data given by the users is encoded. However since this approach used dynamic SQL to encode the data, therefore the method did not prevent the system from the attack.
[12]	Input Validation	Input validation is only at the application or client side and ignores the validation at the server side.
[13]	Dynamic SQL	The data is not properly validated and error messages are not handled consistently.
[14]	Error handling	Proposed signature-based detection to handle error messages. However, this approach did not embed the input validation at both client and server side.
[18]	Dynamic SQL	Validate the input at both client and server. However, SQL injection is still being used and error handling is not properly done.
[21]	Input Validation	Input validation is only at the application or client side and ignores the validation at the server side.
[23]	Parameterized query	This approach did not validate the input data; therefore invalid data might enter the database.
[24]	Input validation	Use dynamic SQL to separate between normal data and malicious data. However, this process still vulnerable to the attack.
[25]	Dynamic SQL	This method uses combined static and dynamic analysis to prevent malicious code produced by dynamic SQL. However, the process will increase the query processing time. The data is not properly validated and error messages are not handled consistently.
[26]	dynamic SQL	Used dynamic SQL in the methodology, therefore not efficient in preventing SQL injection
[27]	Parameterized query	Use Knuth-Morris-Pratt matching algorithm and parameterized query to detect and prevent SQL injection. However, the error messages are not handled properly.
[28]	Parameterized query	Use signature-based approach to prevent the attack. However, the error messages are not handled properly.

Based on the comparative analysis and to overcome the limitations of the existing methods this study proposed a comprehensive method to address the highlighted issues. The existing stored procedure methods is referred and improved by incorporating parameterized queries as a SQL injection prevention mechanism at the database level and developing input validation and error handling as a preventive mechanism at the application level. The proposed method intends to address the issues highlighted in Section IV. The details on the proposed method are discussed in the next section.

VI. AN IMPROVED STORED PROCEDURE WITH PARAMETERIZED QUERY

Currently, most organization used stored procedure as a prevention method at the application level [23][27][28]. Stored procedures are subroutines which contain a set of predefined SQL code that can be saved and reused over and over again when needed. The code is written, compiled and executed before it is being used by the web application to communicate with databases. Frequently used queries can be saved in a stored procedure and can be recalled when requested. Since the location of stored procedure is within the database server, therefore the interaction between the program code and the table containing the data can be avoided [29]. This will improve the security of application program code because it can add another layer of abstraction before interacting with database [19]. With these features, stored procedure is able to provide high security feature to the database.

A parameterized query (or a prepared query statement) is a pre-compiling SQL statement. One or more parameters (or variables) need to embed into the statement for it to be executed. Parameters are sent to the query as soon as the user presses the 'Submit' button or presses the link in the web application. Normally, the existing code for the web application contains dynamic SQL statements when performing a query in the search functions. Due to limitation in dynamic SQL, parameter queries will be used instead, in conjunction with stored procedures. Parameter queries have program code security features that can be used instead of dynamic SQL. Parameters containing embedded user input will not be interpreted as commands to execute, thus will not allow code to be injected by SQL injection. The use of these parameterized queries is intended to replace dynamic SQL queries that can cause vulnerabilities to SQL injection. Application code will be more secured because functional logic is defined first and input parameters are entered after functional logic is processed [2].

Therefore this study proposed a comprehensive method to overcome the three issues highlighted earlier as discussed in Section 4. The proposed method makes an improvement based on these three steps:

1) use stored procedure with parameterized query instead of SQL dynamic,

2) validate the input at both client and server to reduce vulnerabilities to SQL injection. This is to ensure the input parameters sent to the database server do not contain malicious code and to prevent the code from accessing and harm the stored procedure,

3) improve the error handling process by not displaying confidential database information. Since error messages may reveal the limitation of the system to the intruder.

Fig. 2 shows the workflow of the proposed method. The workflow consists of four components which are:

1) the search page where the user enters the data to search,

2) the data validation at the client and the server site,

3) the error message if problems encountered during the process, and

4) the output of the query.

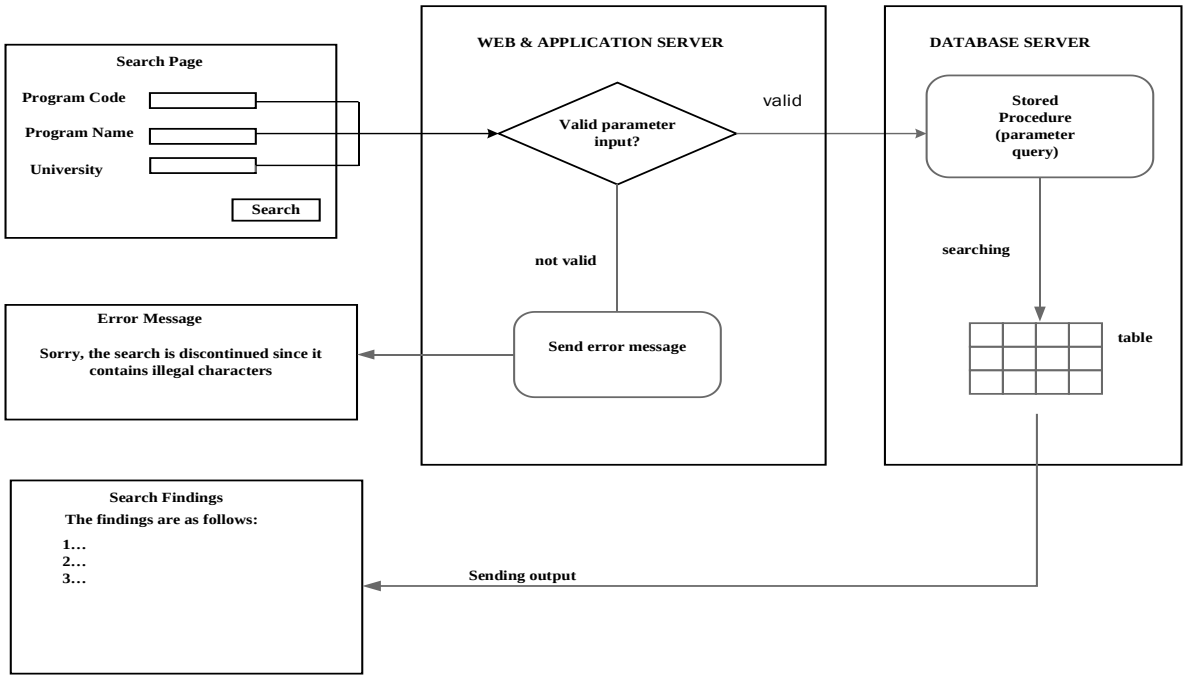

Fig. 2. The Workflow of the Parameterized Stored Procedure Improvement.

```
USE [ProgDB]
GO
/******Object: StoredProcedure [dbo].[search_programmes] ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo] [search_programme]
        @arg_programmecode NVARCHAR(10),
        @arg_programmename NVARCHAR(25),
        @arg_instname NVARCHAR(25),
AS
BEGIN
        DECLARE @sql NVARCHAR(max),
        DECLARE @a NVARCHAR(100),
        DECLARE @b NVARCHAR(100),
        DECLARE @c NVARCHAR(100),
        SET @sql 'SELECT programme_code,
                                autoID
                                inst_name_E,
                                inst_name_ML,
                                inst_name_EL,
                                prog_name_EN,
                                programme_name_E,
                                Category,
                                Status,
                                Valid_date, AA_yes,
                                FROM Q_search where 1=1'
SET @a = ' % ' + @arg_programmecode + ' % '
SET @b = ' % ' + @arg_programmename + ' % '
SET @c = ' % ' + @arg_instname + ' % '
        IF (@arg_programmecode is not NULL)
        SET @sql @sql + ' AND programme_code LIKE ' + ' @a '
        IF (@arg_programmename is not NULL)
        SET (@sql @sql + ' AND programme_name LIKE ' + ' @b '

        IF (@arg_instname is not NULL)
        SET (@sql @sql + ' AND inst_name_E LIKE ' + ' @c'
EXECUTE sp_executesql @sql,
        N '@a NVARCHAR(100) , @b NVARCHAR(100), @c NVARCHAR(100)
',
@a = @a , @


-
```

Fig. 3. The Proposed Method using Parameterized Stored Procedure.

A stored procedure with parameterized query will be constructed that consists of the search function and the four components described above. SQL dynamic is not used in the query. For instance, the user will enter data regarding the program code, program name and university in the search page and the search button is clicked when finished. The input parameter will be validated at both the application and the database site. If the input data is not valid at the application site then an error message will be displayed. If the input data is valid then parameters of the search function will be sent to the stored procedure. The query will search from the table for the answers and displays the result.

The stored procedure that has the search function and the four components used in this study is shown in Fig. 3. This stored procedure consists of validating features and a proper error handling process.

VII. Experimental Design

Since most approaches used dynamic SQL in preventing SQL injection attacks [13][18][25]; therefore this study intends to prove that the proposed method using parameterized query stored procedure is better than the existing approaches. To evaluate the effectiveness of the proposed approach, two experiments are conducted. The purposes for evaluation are to prove that:

1) The stored procedures that have parameterized queries are better than dynamic SQL in terms of preventing databases from SQL injection attacks.

2) The proposed approach has a shorter query processing time when compared the approaches that used dynamic SQL.

To achieve the aims, the experiment is conducted in two versions, which are:

- Experiment 1: the simulation attacks on web applications with dynamic SQL queries, and
- Experiment 2: the simulation attacks on web applications with stored procedures.

In this study, the web application that used as a case study will be represented by a pseudo-version web application. This application is developed to simulate and to prove the existence of web application vulnerabilities to SQL injection and subsequently became a medium for the experiment. A pseudo web application is considered a basic replica of the original web because it is developed with the same program code and functional logic as the original web application. Fig. 4 explains the used of pseudo-version web application in identifying the SQL injection attacks.

Pseudo-version web application is developed and experiment is conduct to simulate the SQL injection attacks. The experiment is run in a computer using the Microsoft Windows 7 Enterprise Edition operating system, 4GB of RAM, Intel (R) Core (TM) i5-2320 CPU @ 3.00GHz processing chip. Pseudo web applications are developed using Microsoft SQL Server 2008 database. The web server operating system used is Windows 2008 R2 with ASP.Net, Coldfusion and Microsoft IIS 7.5 web application technologies. The stored procedures are written in the Transact-SQL language in Microsoft SQL Server.

SQLMap is used in the experiment which works in conjunction with the Python library and Netsparker Trial Edition software. SQLMap is chosen as the SQL injection vulnerability scanner instrument in this study due to its availability as open source software. This software is the most effective and popular among hackers and web application penetration testers [25]. Both experiments used the same test plan and software namely SQLMap, Netsparker and web browser. The query used for these experiments are based on the search query as in the stored procedure in Fig. 3. The setting for these experiments is explained in Fig. 5.

A. Experiment 1

The purpose of experiment 1 is to investigate the capabilities of dynamic SQL and the proposed parameterized stored procedure in preventing SQL injection attacks. To accomplish this experiment, two testing are required.

- Test 1: to evaluate SQL Injection attack on web application with the existence of dynamic SQL queries, and.
- Test 2: to evaluate SQL Injection attack on web application with parameterized stored procedures.

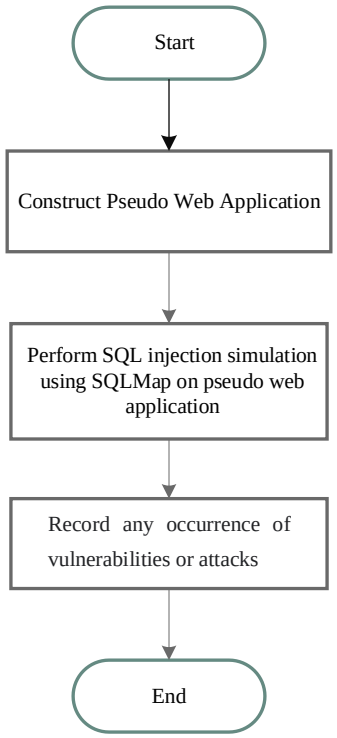

Fig. 4. Process Flow to Identify Web Application Vulnerabilities against SQL Injection.

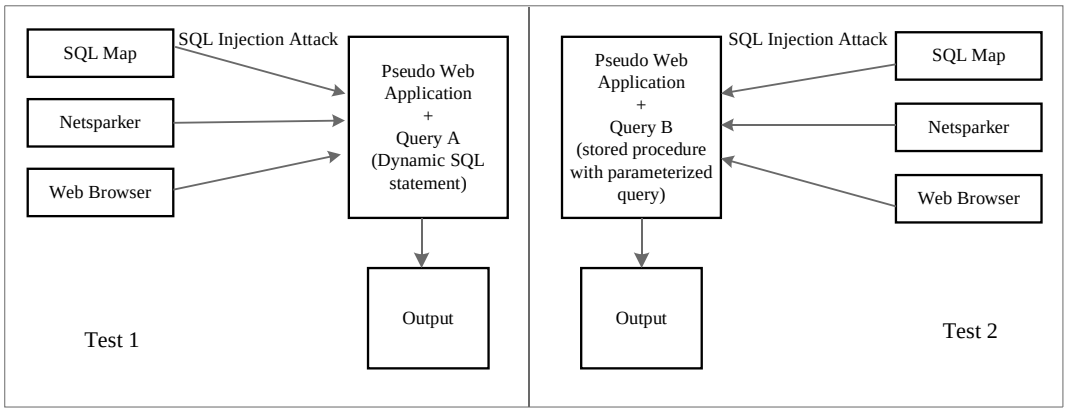

Fig. 5. The Setting for Experiment 1.

SQL which contains a malicious code is injected to SQLMap, Netsparker and web browser separately. SQL injection will penetrates web applications with queries containing dynamic SQL. Web applications that have strong defenses will be able to prevent attacks caused by SQL injection. Upon completion of the experiment, the outputs from these two tests were compared to investigate whether web application with dynamic SQL and stored procedures successful prevent SQL injection attacks. The results of this experiment are displayed in Table III.

TABLE III. THE RESULTS FOR EXPERIMENT 1

Attack	**Experimental Results**	
	Test 1	**Test 2**
SQLMap	Prevention Fail	Successful prevention
Netsparker	Prevention Fail	Successful prevention
Web Browser	Prevention Fail	Successful prevention

It is found that web applications with dynamic SQL is unable to prevent the attack cause by SQL injection when using SQLMap, Netsparker or web browser as stated in Test 1. The same instruments are used to test the web applications equipped with parameterized query, error handling and input validation. However when using the proposed approach, the web application able to prevent the attack. This proves that the proposed parameterized stored procedure is able to prevent web applications from SQL injection attacks.

B. *Experiment 2*

The purpose of Experiment 2 is to evaluate the performance of the proposed method and dynamic SQL in terms of time processing. This study used two evaluation instruments, namely

- Microsoft SQL Server Client Statistics and
- SQLQueryStress.

Client Statistics is a facility available in Microsoft SQL Server while SQLQueryStress is a standalone tool available as open source.

1) Evaluation using microsoft SQL server client statistics: This evaluation use MS SQL Server Client Statistics as an instrument to investigate the amount of data transmitted from server to client side. The intention is to evaluate whether SQL statement contributes to high traffic workload. The client execution time and processing time will be recorded during this evaluation.

a) Measuring client execution time: Client execution time is the cumulative amount of time used at the client side to execute query. To obtain the average execution time, ten trials were performed. In this evaluation, Test 1 represents the pseudo-web application which uses dynamic SQL during the trial, while Test 2 represents web-application which uses the proposed parameterized stored procedure. The output from this evaluation is displayed in Fig. 6.

Fig. 6 states that for each execution, Test 1 generates high execution time when compared with Test 2. This is due to a stored procedure which located at the database server. The SQL query is executed first before the web application submits the request. As such, the client is able to shorten the execution time by calling SQL queries that are already executed at the database server. In contrast, dynamic SQL are executed at runtime when the requests are submitted by the user. However, the query execution can be influenced by various factors. Such as, network traffic conditions as well as web and database server workloads. This will increase client execution time during query processing when using dynamic SQL. As the results show, the proposed parameterized stored procedure is better than dynamic SQL in terms of processing time.

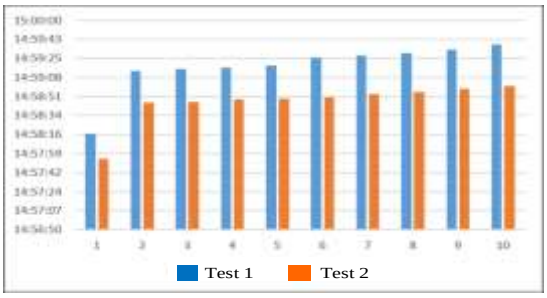

Fig. 6. Query Processing Time at the Client Side.

b) Measuring Time Statistic: Time statistics provide information on the amount of time used during query processing at the client side versus the time used to wait for the server. Three metrics that can be used to measure time statistics namely client processing time, server response time and total execution time.

- Client processing time- to measure query processing time at the client side.
- Wait-time on server replies - is the time spent by the client waiting to receive the response from a database server after submitting the request.
- Total execution time - the time spent by the system to execute the query, including the time spent waiting for the server to respond. Using the same instrument, the results from this experiment are shown in Table 4. Comparison of time statistics between Test 1 and Test 2.

TABLE IV. THE OUTPUT FOR TEST 1 AND TEST 2

Evaluation Criteria	Test 1 (ms)	Test 2 (ms)
i. client processing time	3.5	1.9
ii.server response time	40.2	39.6
iii.total execution time (i+ii)	43.5	41.5

As can be seen in Table IV, web applications that use stored procedures are able to process queries at the client side in a shorter time. Also in the second and third evaluation criteria, the applications with stored procedure are able to improve query processing time at the server and total execution time both at the client and server when compared with the applications with dynamic SQL. This is due to the query in the stored procedure is executed first before the web application submits the request. Using this improvement method, the time taken to process the queries will be reduced.

2) Evaluation using SQL query stress: This evaluation uses SQLQueryStress software and used as a benchmark to investigate the performance of the query when given heavy workload. The intention is to investigate the impact of SQL queries (whether dynamic SQL queries, parametric queries or stored procedures) towards system performance. The differences in the structure of dynamic SQL queries and stored procedures make the query executed in different ways. Therefore, it is important to know the impact of both SQL queries on the system performance. Based on the result in Table V, it is found that the time taken by Test 2 is less than the time taken by Test 1.

The results state that Test 2 has a logic reading of 1547 compared to 1620 in Test 1. SQL queries that have low logic readings were more efficient than SQL queries that produced high logic readings. This is because high logic readings have negative implications to system memory. In other words, high logic readings place heavy workload to computer systems. The results show that query embedded in stored procedures produce less workload than dynamic SQL when using the same data hence improve query performance. This is due to the query is processed earlier at the database server rather than at the runtime. Thus, it will reduce the elapsed time, the CPU time, the actual processing time and the client time as well.

TABLE V. COMPARISON OF SQL QUERYSTRESS CRITERIA ON WEB APPLICATIONS USING DYNAMIC SQL QUERIES (TEST 1) AND STORED PROCEDURES (TEST 2)

Criteria (average)	**Test 1**	**Test 2**
elapsed time (ms)	1.3600	0.7350
CPU seconds (ms)	0.0996	0.0167
actual seconds (ms)	0.2834	0.0401
client seconds (ms)	0.1009	0.0475
logical reads	1620.0	1547.0

VIII. CONCLUSIONS

This study successfully proposed a new comprehensive method using an improved parameterized stored procedure to overcome the three issues highlighted in preventing web application from SQL injection attack. These three issues are i) improper use of dynamic SQL, ii) lack of input validation process and iii) inconsistent error handling. Unfortunately, the existing approaches did not properly address these issues. Hence they are not able to prevent SQL injection attacks at both the application and database side. In this study, a stored procedure is constructed that consists of a comprehensive method to address these three issues which are built in the code. To prove the effectiveness, the proposed method was evaluated through three approach of attack simulation, namely using SQLMap software, Netsparker and web browser. The experiments conclude that SQL injection does not successfully penetrate web applications and databases when the proposed method is implemented hence able to overcome the limitations faced by the existing methods. This indicates that the SQL injection prevention method developed has successfully prevents SQL injection from occurring. The proposed method is also evaluated from the perspective of time used and its impact on the overall system. Evaluations using Microsoft SQL Server Client Statistics and SQLQueryStress software have concluded that although there are slight differences in time processing, the proposed method uses a shorter query processing time when compared with dynamic SQL. It can be concluded that the SQL injection prevention method used does not generate high overhead that may lead to high response time. This study can be further improved by focusing the prevention of SQL injection in network systems. There are various factors to be considered such as the types of server used, network traffic and the attacks from various sources. More efforts are needed and further enhancements are required to improve database security.

ACKNOWLEDGMENT

This research was sponsored by the Research Incentive Grants (Grant No. GGP-2019-024), Centre for Software Technology and Management (SOFTAM) of Faculty of Information Science and Technology, National University of Malaysia (UKM).

REFERENCES

[1] A.Jamil and Zawiyah Mohammad Yusof, "Information Security Governance Framework of Malaysia Public Sector", Asia-Pacific Journal of Information Technology and Multimedia, vol. 7 no. 2, 2018, pp.85 – 98.

[2] L.Ma, D.Zhao, Y.Gao and C. Zhao, "Research on SQL Injection Attack and Prevention Technology Based on Web", International Conference on Computer Network, Electronic and Automation (ICCNEA), Xi'an, China, 2019, pp. 176-179.

[3] N. Singh, M. Dayal, R. S. Raw and S. Kumar, "SQL injection: Types, methodology, attack queries and prevention," 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2016, pp. 2872-2876.

[4] Z. C. S. S. Hlaing and M. Khaing, "A Detection and Prevention Technique on SQL Injection Attacks", IEEE Conference on Computer Applications (ICCA), Yangon, Myanmar, 2020, pp. 1-6.

[5] H. Gupta, S. Mondal, S. Ray, B. Giri, R. Majumdar and V. P. Mishra, "Impact of SQL Injection in Database Security", International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), Dubai, United Arab Emirates, 2019, pp. 296-299.

[6] M.Malik, and Patel, T., "Database Security - Attacks and Control Methods", International Journal of Information Sciences and Techniques, 6(1/2), 2016, pp.175–183.

[7] H. Gaikwad, Bhavesh B. Shah and Priyanka Chatte, "SQLi and XSS Attack Introduction and Prevention Technique", International Journal of Computer Applications (0975 – 8887) May 2017 volume 165 – No.2, 23-27.

[8] J. Clarke, SQL Injection Attacks and Defense, Second Edition, 2012. Syngress, pp.1–473.

[9] P. S. P. Pullagura and Gokilavani, "Defeating SQL Injection on Preventing Run Time Attacks", International Journal Of Science & Technoledge, 2(5), 2014, pp. 93–96.

[10] Kindy, D. A. & Pathan, A. K.. A Detailed Survey on various aspects of SQL Injection in Web Applications : Vulnerabilities , Innovative Attacks and Remedies. International Journal of Communication Networks and Information Security, 5(2), 2013, pp.80–92.

[11] Q. Li, W. Li, J. Wang and M. Cheng, "A SQL Injection Detection Method Based on Adaptive Deep Forest," in IEEE Access, vol. 7, 2019: pp. 145385-145394,

[12] V. Dwivedi , Himanshu Yadav and Anurag Jain, "SQLas: Tool to Detect And Prevent Attacks in PHP Web Applications", International Journal of Security, Privacy and Trust Management (IJSPTM) Vol 4, No 1, February 2015, pp.21-30.

[13] G. Deepa and Thilagam, P. S.. "Securing Web Applications from Injection and Logic Vulnerabilities: Approaches and Challenges", Information and Software Technology, 74, 2016, pp.160–180.

[14] A.Jumaa and Omar, A.. "Online Database Intrusion Detection System Based on Query Signatures", Journal of University of Human Development, 3(1), 2017, pp.282–287.

[15] M. Rami and F. Jnena, "Modern Approach for WEB Applications Vulnerability Analysis", Master of Science in Computer Engineering. The Islamic University of Gaza Deanery of Graduate Studies Faculty of Engineering Computer Engineering Department, 2013.

[16] M.Amirulluqman Azman, Mohd Fadzli Marhusin and Rossilawati Sulaiman, "Machine Learning-Based Technique to Detect SQL Injection Attack", Journal of Computer Science, 17 (3), 2021, pp.296-303.

[17] J.Minhas and Kumar Raman, "Blocking of SQL Injection Attacks by Comparing Static and Dynamic Queries", International Journal of Computer Network and Information Security; 5(2), Feb 2013, pp.1-9.

[18] M.Amin Mohd Yunus, Muhammad Zainulariff Brohan and Nazri Mohd Nawi. "Review of SQL Injection: Problems and Prevention". International Journal On Informatics Visualization, vol 2, 2018, No 3 – 2.

[19] S. Choudhary and Nanhay Singh.. Safety "Measures and Auto Detection against SQL Injection Attacks", International Journal of Engineering and Advanced Technology (IJEAT), Volume-9 Issue-2. 2019, pp. 2827 – 2833.

[20] P. Sadotra and Chandrakant Sharma. SQL "Injection Impact On Web Server And Their Risk Mitigation Policy Implementation Techniques: An Ultimate Solution To Prevent Computer Network From Illegal Intrusion", International Journal of Advanced Research in Computer Science,Volume 8, No. 3, March – April 2017, pp. 678-686.

[21] K.G. Vamshi, V. Trinadh, S. Soundabaya, and A. Omar, "Advanced Automated SQL Injection Attacks and Defensive Mechanisms", in Annual Connecticut Conference on Industrial Electronics, Technology & Automation (CT-IETA), 2016, p. 1-6.

[22] A.Alazab and Ansam Khresiat, "New Strategy for Mitigating of SQL Injection Attack", International Journal of Computer Applications 154(11), 2016, pp.1-10.

[23] M.Horner and Hyslip, T.. "SQL Injection: The Longest Running Sequel in Programming History", Journal of Digital Forensics, 12(2), 2017, article 10.

[24] R. Mohamed Thiyab, Musab A. M. Ali, Farooq Basil and Abdulqader, "The impact of SQL injection attacks on the security of databases", Proceedings of the 6th International Conference of Computing & Informatics, 2017, pp. 323-331.

[25] I.Lee, 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, 2012, Pages 58-68.

[26] S. Nanhay, D. Mohit, R.S. Raw, and K. Suresh, "SQL Injection: Types, Methodology, Attack Queries and Prevention", in 3rd International Conference on Computing for Sustainable Global Development (INDIACom), 2016, p. 2872 – 2876.

[27] O.C. Abikoye, Abubakar, A., Dokoro, A.H. et al. "A novel technique to prevent SQL injection and cross-site scripting attacks using Knuth-Morris-Pratt string match algorithm", EURASIP J. on Info. Security, 2020, 14.

[28] S. Choudhary, Arvind Kumar and Anil Kumar, "A Detail Survey on Various Aspects of SQLI", International Journal of Computer Applications, 161(12), 2017, pp.34-39.

[29] A. Zhu and Wei Qi Yan, "Exploring Defense of SQL Injection Attack in Penetration Testing", International Journal of Digital Crime and Forensics, 9(4), 2016, pp. 62-71.