The Multi-Tier Architecture for Developing Secure Website with Detection and Prevention of SQL-Injection Attacks

Praveen Kumar
Department of Software Systems,
Samrat Ashok Technological Institute, Vidisha, M.P. INDIA

ABSTRACT
SQL injection is an attack methodology that targets the data residing in a database. The attack takes advantage of poor input validation in code and website administration. SQL Injection Attacks occur when an attacker is able to insert a series of SQL statements into a 'query' by manipulating user input data into a web-based application, an attacker can take advantages of web application programming security flaws and pass unexpected malicious SQL statements through a web application for execution by the back-end database. This paper proposes a novel specification-based methodology for the prevention of SQL injection Attacks. The two most important advantages of the new approach against existing analogous mechanisms are that, first, it prevents all forms of SQL injection attacks; second, Current technique does not allow the user to access database directly from the database server.
Our proposed framework for building secure and anti-theft web applications is consisting of four stages. In each stage we analyze the inputted data taken from the user and make a decision, whether that is suspected or not.

Keywords*:* SQL Injection, Avoidance, SQLIA Prevention, SQLIA Detection, SQL Attacks.

1. INTRODUCTION
For past years many organizations provide their services on the Internet. For that they keep the information regarding their customers, their partners which is quite sensitive in that they maintain the database driven web application. SQL Injection is a type of injection or attack in a Web application, in which the attacker provides Structured Query Language (SQL) code to a user input box on a Web form to gain unauthorized and unlimited access. The attacker's input is transmitted in an SQL query in such a way that it will form an SQL code [1], [2].

SQL injection is a type of security attack, which attacks web applications that are using database services. There are three forms of SQL injections:

1. Incorrectly filtered escape characters
2. Incorrect type handling
3. Blind SQL injection

The first form of SQL injection is incorrectly filtered escape characters which occur when user input is not filtered for escape characters and then transferred to the SQL statements. This results in the possible manipulation of the statements made on the database end-user applications.

The second form of SQL injection is the incorrect type handling. This form of SQL injection occurs when a user supplied field is not strongly typed or not to control the type constraints. This would occur when a numeric field is to be used in SQL instruction, but the programmer does not check to confirm that the user input is numeric.

The third form of SQL injection is the Blind SQL injection, a blind SQL injection is used when a web application is vulnerable to SQL injection, but the results of the injections are not visible to the attacker. About the vulnerability cannot be one that displays the data, but will appear differently depending on the outcome of legal logic is injected into the SQL statements called from this page.

Aim of SQL injection is to query the database a manner that was not the intent of the application programmer. There are several techniques used in SQL injection. Most of them use SQL statement in different SQL injection techniques. Increased dependence on web applications significantly and use in the activities of our daily lives grows in the number and level of attacks that target them.

2. SECURITY ISSUES: THREATS TO WEBSITE
Due to the increased use of online and automated processes, huge bulks of sensitive and critical data are being handled by the web applications. As the stakes on the information and data stored by the portals become higher, so does the sophistication of hackers. Developers and hackers are racing against each other. Developers try to make the web application secure from the threats and the hacker wish to find the loophole, so that it can steal or damage the application or data. Security threats could be with the intent of stealing confidential information, causing deliberate damage, prove capability or simply for the thrill of doing something which most others cannot do.

In today's scenario, IT enabled services are very common in commercial as well as in the government sector, which results number of eCommerce websites. These websites helps the users to make personal account there for online transactions, or to store their confidential data. Almost all the organization builds their personal servers to which all the organizational records and data are managed. In this way, the data and other web content become very precious to any organization or individual.

Therefore, a developer needs to take all the precautions to secure the organization's data by avoiding or preventing all the security threats to the website. Usually following are the most common threats to a website:

1. Brute Force: the attacker simply keeps on guessing and trying various combinations, most often using an automated script, to gain unauthorized access to a system.

2. Cross-Site Scripting: Cross-site scripting (XSS) is a type of website security vulnerability which allows attackers to inject client-side script in web pages viewed by users of the website. Various input controls like Rich Text Editor, etc. which allow users to add html tags as part of input, from the root cause behind these attacks.

3. Insufficient Anti-Automation: when a web site permits an attacker to automate a process that should only be performed manually. A certain web site functionalities should be protected against automated attacks. Left unchecked, automated robots (programs) or attackers could repeatedly exercise web site functionality attempting to exploit or defraud the system. An automated robot could potentially execute thousands of requests a minute, causing potential loss of performance or service.

4. Denial of Service: In a denial-of-service attack, an attacker may block access of a website to its legitimate users. Most of the times, this is done by flooding the web server hosting the website with repeated requests for Web pages, through script automation.

5. Network Sniffers: Network sniffer can list all of the network packets in real-time from multi network card (Include modems, ISDN, ADSL, etc.) and can support capturing packets based on the applications (Socket, TDI, etc.). An attacker can observe the traffic of the application. It is easy to learn and simple to use. Network sniffer has plug-ins for different protocols such as Ethernet, IP, TCP, UDP, PPPOE, HTTP, FTP, WINS, PPP, SMTP, POP3 and so on. Network sniffers can be used to identify sensitive information like credit card information and details about the systems involved in processing credit card transactions.

6. SQL Injections: SQL injection is an attack in which malicious code is inserted into strings, which are later passed to the database server for parsing and execution. This attack can be applies to any page which accepts user input to capture data or query parameters to dynamically render the content on the web page.

3. VULNERABILITIES DUE TO DATABASE

All know it that, websites and internet are the cheapest and easiest way for promotion and publicity. It provides an international platform, to convey our message to all viewers. Other than this, the websites also used for its incredible business capabilities, like shopping websites and social websites. Therefore, Most of the website is driven by their databases, which enables them to show the dynamic data content on the user requests. Hence, the database is very precious for any organization, as it contains the valuable and confidential data. While, the website is non-other than a medium that represent the data contents in a presentable form and allows the web users to communicate with the database in a legal way.

One of the major vulnerabilities to a website is the vulnerability due to the databases. This may be the loss of data, damage to the database, unauthorized access, bypassing the authentication mechanisms or denial of services. These actions can be done in the following ways:

Injection through cookies: Cookies are those files that contain state information that are generated by Web applications and stored on the client machine. When a client returns to a Web application, cookies can be used to restore the client state information. Since the client has control over the storage of the cookie, a malicious client could tamper with the cookie's contents. If a Web application uses the cookie's contents to build SQL queries, an attacker could easily submit an attack by embedding it in the cookie [3].

Injection through user input: The attackers inject SQL commands by providing suitably crafted user input. In most SQLIAs that target Web applications, user input typically comes from a form

Submissions that are sent to the Web application via HTTP GET or POST requests [8]. Web applications are generally able to access the user input contained in these requests as they would access any other variable in the environment.

Injection through server variables: Server variables are a collection of variables that contain HTTP, network headers, and environmental variables. Web applications use these server variables in a variety of ways, such as logging usage statistics and identifying browsing trends. If these variables are logged to a database without sanitation, this could create an SQL injection vulnerability.

Second-order injection: Second-order injections are not trying to cause the attack to occur when the malicious input initially reaches the database. Instead, attackers rely on knowledge of where the input will be subsequently used and craft their attack so that it occurs during that usage.

4. SQL INJECTIONS CLASSIFICATION

Database applications have become a core component in control systems and their associated record keeping utilities. Traditional security models attempt to secure systems by isolating core software components and concentrating security efforts against threats specific to those computers or software components.

Type of Attack	**Attack Intent**
Tautologies	• Authentication • Identifying Injectable Parameters • Extracting Data
Logically Incorrect Queries	• Identifying Injectable Parameters • Performing Database Fingerprinting • Extracting Data
Union Query	• Bypassing Authentication • Extracting Data
Piggy Backed Queries	• Extracting Data • Adding or Modifying Data • Performing Denial of Service • Executing Remote Commands
Stored Procedures	• Performing Privilege Escalation

	• Performing Denial of Service • Executing Remote Commands
Blind Injection	• Identifying Injectable Parameters • Extracting Data • Determining Database Schema
Timing Attacks	• Identifying Injectable Parameters • Extracting Data • Determining Database Schema
Alternate Encodings	• Evading detection

5. PROBLEM DEFINITION

The SQL injections are the major threats to any website as it is concerned to the database, which is the valuable to any organization. The SQL injections are the top most of the threats for web applications security and database is the most valuable asset for any organization. The problems of SQL injection attacks are increasing very rapidly because they are easy to employ by putting the malicious code etc.

1- To detect and prevent the SQL injection attacks, we need an intelligent mechanism. That must be easily deployable, and provide good performance in analyzing the malicious code.
2- To Implement a sanitizer tool that decides whether to send the data to database manager or not.
3- To have a mechanism that does not affect the system performance by engaging the CPU for more time.
4- To propose an architecture that must follow by the web developers for the building of a secured website.
5- To propose the hierarchy to classify the inputs that helps to identify in sanitizations process. Improper Neutralization of Special Elements used in an SQL Command.

6. RELATED WORK

Although the SQL injection is not a new field of research but it is one of the most popular fields for researchers, Contribution to this field such as filtering, information flow analysis, penetration testing, and defensive coding, can detect and prevent a subset of the vulnerabilities that lead to SQL Injections Attacks. Some of the researches we studied for our references are shown below:

William G. J. Halfond et al.'s Scheme- [2] - This approach works by combining static analysis and runtime monitoring while SAFELI – [5] proposes a Static Analysis Framework in order to detect SQL Injection Vulnerabilities. SAFELI framework aims at identifying the SQL Injection attacks during the compile-time. This static analysis tool has two main advantages. Firstly, it does a White-box Static Analysis and secondly, it uses a Hybrid-Constraint Solver. For the White-box Static Analysis, the proposed approach considers the byte-code and deals mainly with strings. For the Hybrid-Constraint Solver, the method implements an efficient string analysis tool which is able to deal with Boolean, integer and string variables.

Thomas et al.'s Scheme - Thomas et al., in [6] suggest an automated prepared statement generation algorithm to remove SQL Injection Vulnerabilities. They implement their research work using four open source projects namely: (i) Net-trust, (ii) trust, (iii) WebGoat, and (iv) Roller. Based on the experimental results, their prepared statement code was able to successfully replace 94% of the SQLIVs in four open source projects.

Ruse et al.'s Approach - In [7], Ruse et al. Propose a technique that uses automatic test case generation to detect SQL Injection Vulnerabilities. The main idea behind this framework is based on creating a specific model that deals with SQL queries automatically. Adding to that, the approach identifies the relationship (dependency) between sub-queries. Based on the results, the methodology is shown to be able to specifically identify the causal set and obtain 85% and 69% reduction respectively while experimenting on few sample examples.

Ali et al.'s Scheme - [8] adopts the hash value approach to further improve the user authentication mechanism. They use the user name and password hash values SQLIPA (SQL Injection Protector for Authentication) prototype was developed in order to test the framework. The user name and password hash values are created and calculated at runtime for the first time the particular user account is created.

SQL-IDS Approach - Kemalis and Tzouramanis in [10] suggest using a novel specification-based methodology for the detection of exploitations of SQL injection vulnerabilities. The proposed query-specific detection allowed the system to perform focused analysis at negligible computational overhead without producing false positives or false negatives.

SQLIA Prevention Using Stored Procedures - Stored procedures are subroutines in the database which the applications can make call to[12] . The prevention in these stored procedures is implemented by a combination of static analysis and runtime analysis. The static analysis used for commands identification is achieved through the stored procedure parser and the runtime analysis by using a SQL Checker for input identification.

Similarly various works like SQLrand Scheme - SQLrand approach [11], Parse Tree Validation Approach - Buehrer et al. [13] adopt the parse tree framework, Dynamic Candidate Evaluations Approach - In [14], Bisht et al. Propose CANDID and Roichman and Gudes's Scheme – [9]. All these approaches tried to solved out the problems associated with the SQL injections and other vulnerablilites.

7. PROPOSED ARCHITECTURE

The proposed technique consists of four-stage security mechanism, which are as follows:-

1. Validations stage: - This stage works on client side "web page at browser"; it detects the use of special characters with the help of regular expressions. This technique is called validations. With this mechanism, we are able to protect the web page controls like (text boxes) from any kind of special character, which is meaningful to the SQL. The only drawback of this stage is that, it cannot handle the attacks from the query strings, cookies etc.
2. Data filtration Stage: - This stage basically works on the code written behind the design page. This stage detects the malicious SQL code, inject by various other techniques like query strings, URL's. The data filtration stage analyzes the data before passing on to the database.

3. Dynamic Guard Stage (Sanitizer): - Parameterized query is parameterized database access API provided by development platform such as Prepare Statement in Java or SQL Parameter .NET. Instead of composing SQL by concatenating string, each parameter in a SQL query is declared using placeholder and input is provided separately. The sanitizer classify the inputs as follows-

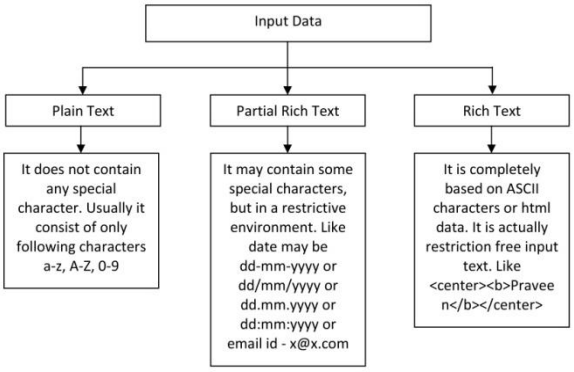

Figure 1 Input Classification based on their nature and types.

4. Parameterized Check Stage: - Parameterized queries keep the query and the data separate through the use of placeholders known as "bound" parameters. This helps in preventing SQLIA by not allowing the structure of the query to be altered; rather it merely "fills in" the input parameters into their positions and keeps the rest of the query structure intact. Since a majority of the SQLIA techniques relies on altering the query structure for injection attacks, this serves as a very effective combat technique.

   In fig. 2, It is clearly seen that, how we can monitor the data at different stages during the working on GET & POST methods.

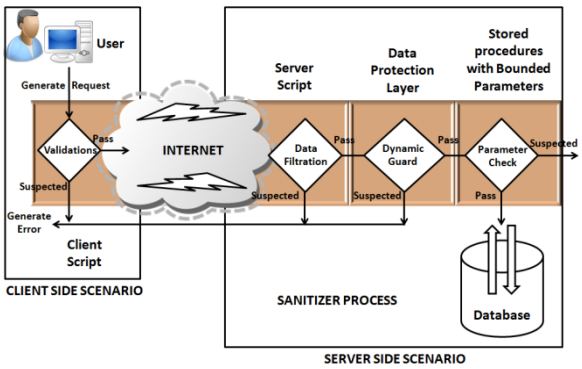

Figure 2 : Proposed Architecture of our plan fro SQL Injections

Following is the algorithm, used in the proposed methodology for the detection and prevention of SQL injections.

In fig. 3, The simple idea to implement the SQL injection detection mechanism is to check each & every input before passing it to the database. For this, we need to perform an analysis that defines us the suspected inputs for different data input fields. Therefore, Based on this analysis we can easily identify the nature of inputs, whether it is an RTF input or non-RTF input.

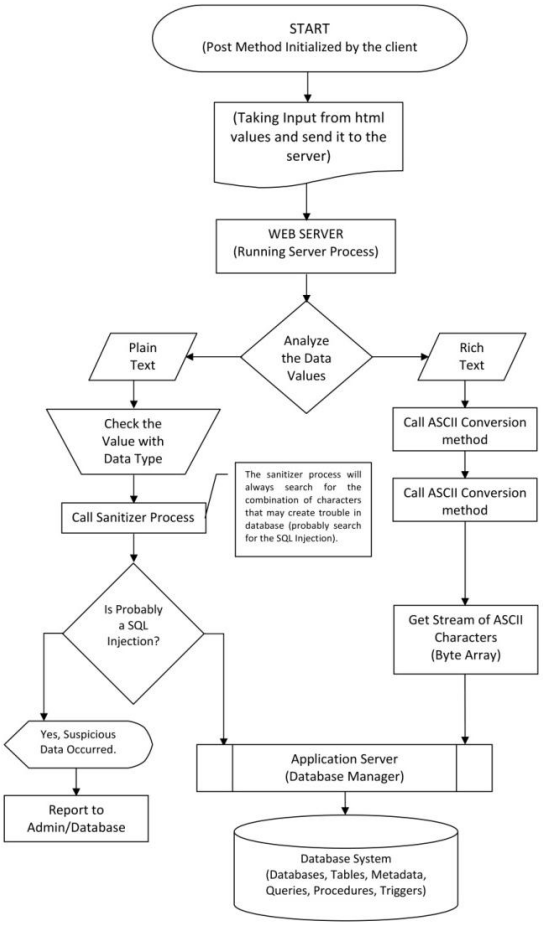

Figure 3: Algorithm for SQL Injection Detection & Prevention

8. IMPLEMENTATION

The complete implementation of the proposed work carried out using the Microsoft .Net framework 3.5 with Visual Studio 2008, SQL Server 2008, IIS Server and a vulnerability scanner tool Netsparker. Through which we scan our own developed website for any vulnerability threat and SQL attacks. The demonstrated website works in two modes at every instance. Whether its login or search, inserting or updating, etc. Following are the two modes:

1. Secure Mode: - In this mode the website works under the guidance of Sanitizer, which sanitizes the inputs and always looking for any intrusion or malicious code in the input. The development of sanitizer leads to a mechanism that helps to decide whether the input data is legitimate and does not contain any malicious code.

   In this mode, the website also works in different tiers. Each tier contains a level of attack detection. For example, the client levels identified the input type and according to that apply the regular expression as the input validation.

   At server side, the website analyses the input data and sends to the database stored procedure, where execution of SQL commands performs and the desired data retrieved.

At data access layer, the website received the data from the internal web pages, and then it uses the sanitization process to make decision on input data before sending to the database program.

The most important feature of our proposed plan for detecting & preventing the SQL injections is that it never overhead the data in the database. It can be applied to any existing web model without performing any alteration in database schema.

Simple Login Query-

```
SELECT username, password FROM tbl_userinfo
WHERE name = 'praveen' AND pass='praveen'
```

Simple Login Query- Stored Procedure

```
CREATE PROCEDURE [dbo].[splogin]
@username nvarchar(20),
@password nvarchar(20)
AS
BEGIN
SET NOCOUNT ON;
SELECT username, password FROM tbl_user
WHERE username = @username AND
password=@password;
END
```

The above stated query can be easily hacked by the attackers. As there is no mechanism for trouble shooting.

Proposed Login Query: Stored procedure

```
CREATE PROCEDURE [dbo].[spsecurelogin]
@username nvarchar(20),
@password nvarchar(20)
AS
BEGIN
SET NOCOUNT ON;
SELECT
        username, password
FROM
        tbl_user
WHERE
        hashbytes('SHA',username)       =
        hashbytes('SHA',@username)
AND
        hashbytes('SHA',password)       =
        hashbytes('SHA',@password);
END
```

The above stated query is free from any malicious code, in-fact if user somehow transmits malicious data to the SQL procedure; our proposed SQL query will compute the Hash values with "SHA" (128 Bits) then it will be compared with database. Therefore, due to applying operation of hash values the input values are sanitizes completely and guarantees you solution in legal way.

Similarly, with the handling of query strings:

The QueryString collection is used to retrieve the variable values in the HTTP query string. The HTTP query string is specified by the values following the question mark (?), like this:

```
Response.Redirect
("admin.aspx?adm_name=prav")
```

The line above generates a variable named adm_name with the value "prav". The Query strings are also generated by form submission, or by a user typing a query into the address bar of the browser.

While at the redirected page, this query string handled by the code for further use like:

```
Dim admin_name as String
admin_name =
Request.QueryString("adm_name")
```

hence, it is the duty of the application programmer to check the validity of this query string value and make sure that, there is no malicious characters or code in the value.

```
IsValid(admin_name)        'Boolean type function
Or
PT_Sanitizer(admin_name) 'Boolean Type Func
```

The above function is uses by our proposed system, which sanitize the admin name and help us to decide whether to send the data to database or not.
The IsValid Code looks like:

```
Public Function IsValid(ByVal str As String) As
Boolean
    Dim rx As New Regex
        ("[0-9a-zA-Z$%^&amp;amp;*()_+!]{3,20}")
    Dim bool As Boolean = rx.IsMatch(str)
    Return bool
  End Function
```

While, the plain text sanitizer simply checks the presence of any trouble making characters.

The PT_Sanitizer code looks like:

```
Public Function PT_Sanitizer (ByVal Str As String) As Boolean
    Dim sb As Byte() = System.Text.ASCIIEncoding.ASCII.GetBytes(Str.ToCharArray())
    Dim i As Integer = 1

    While i <= Str.Length - 1
      If (sb(i) = 32) Or (sb(i) >= 65) And
                (sb(i) <= 90) Or (sb(i) >= 97) And
                (sb(i) <= 122) Or (sb(i) >= 48) And
                (sb(i) <= 57) Then
        i = i + 1
        If i = Str.Length - 1 Then
          Return True                    'Valid Query
          Exit While
        End If
      Else
        Return False          'Injection Query
        Exit While
      End If
    End While
  End Function
```

2. Un-Secure Mode:-

The unsecure mode in the demonstrating website is never uses any SQL injection detection or prevention strategy, it simply pass the data from the HTML input to the database.

9. EXPERIMENTAL RESULT & ANALYSIS

Total Request	Injections Detected	Normal Queries
2420	1635	785
125076	32697	92379

30867	21087	9780
1086	608	478

Figure 4 : Result obtained by using the vulnerability scanner

While working with demonstrating website, by using the vulnerability scanners and concurrent users, we successfully generate the above number of request in four different iterations. The proposed implemented system gives the appropriate results by analysing the inputs provided to them and finally output the counts of the Valid & Injected queries.

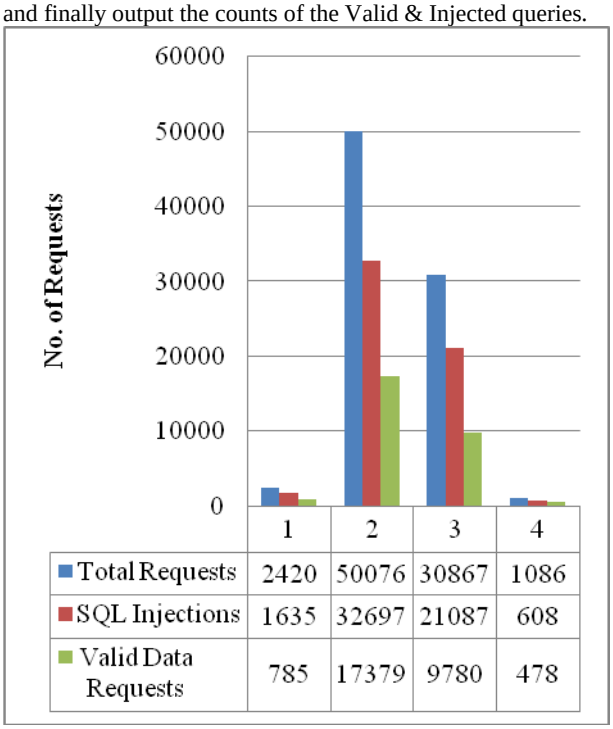

Figure 5 : Graphical representation of result

Following is comparative study of existing SQL detection techniques with the proposed technique.

Technique	Tautology	Illegal Structured	Piggybacking	Union	Stored Procedure	Inference	Alternate Encoding
PROPOSED	Y-D-P	Y-D-P	Y-D-P	Y-D-P	Y-D-P	Y-D-P	Y-D-P
AMNESIA[2]	Y-D	Y-D	Y-D	Y-D	-	Y-D	Y-D
SQL GUARD[13]	Y-D	Y-D	Y-D	Y-D	-	Y-D	Y-D
SQL-IDS[10]	Y-D	Y-D	Y-D	Y-D	Y-D	Y-D	Y-D
Hash + Salt[18]	Y-P	-	Y-P	-	-	-	-

Figure 6: Comparison with existing system for SQL injections detection & prevention

Y- Yes, D- Detection, P-Prevention

10. CONCLUSION & FUTURE WORK

In this paper, we have concentrated on the specific area of SQL injection. Though this is a narrow subject, We believe that this area is in need of further investigation, mainly because of two reasons: first, we can not be certain that we have compiled a definite list of all components that could be taken into consideration. Secondly, SQL injection attacks are most likely to evolve and new vulnerabilities will be found, together with new countermeasures to deal with them. Since many hacking sites are available on the web, and since attack methods are well described and distributed between hackers, we believe that information about new attack methods should continuously be surveyed and new countermeasures should be developed.

According to OWASP's Ten Most Critical Web Application Security Vulnerabilities [16], many SQL injection-related issues are among the most harmful threats to web applications. Since we have in this thesis only covered SQL injection aspects, we would like to suggest that further studies should be

made on other threats to related security issues, especially such that relate to application security. The reason is that several authors have mentioned that organizations spend most security resources on operating system and
network level security [15, 16, 17], and not enough on application layer security. If further studies will be made on application layer security issues, and particularly on web application, it would be possible to compile results from all of these into general security guidelines, which could be used in developing more secure web applications.

One of our goals in this paper was to increase the level of security awareness among organizations regarding web applications, especially towards SQL injection threats. We hope that further surveys in this area and in related web application subjects will help achieving that goal, so that hopefully security standards will be implemented and countermeasures built into applications during development. Ultimately, organizations will use a proactive approach towards application layer security, which will then be an indispensable part of web applications.

11. REFERENCES

[1] A Tajpour, A., Masrom, M., Heydari, M.Z., and Ibrahim, S., SQL injection detection and prevention tools assessment. Proc. 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT'10) 9-11 July (2010), 518-522

[2] L Halfond W. G., Viegas, J., and Orso, A., A Classification of SQL-Injection Attacks and Countermeasures. In Proc. of the Intl. Symposium on Secure Software Engineering, Mar. (2006).

[3] M. Dornseif. Common Failures in Internet Applications,May2005.http://md.hudora.de/presentations/2005-common-failures/dornseif-common-failures-2005-05-25.pdf.

[4] C. A. Mackay. SQL Injection Attacks and Some Tips on How to Prevent Them. Technical report, The Code Project, January 2005. http://www.codeproject.com/cs/database/SqlInjectionAttacks.asp.

[5] X. Fu, X. Lu, B. Peltsverger, S. Chen, K. Qian, and L. Tao. A StaticAnalysis Framework for Detecting SQL Injection Vulnerabilities,COMPSAC 2007, pp.87-96, 24-27 July 2007

[6] S. Thomas, L. Williams, and T. Xie, On automated prepared statement generation to remove SQL injection vulnerabilities. Information and Software Technology 51, 589–598 (2009).

[7] M. Ruse, T. Sarkar and S. Basu. Analysis & Detection of SQL Injection Vulnerabilities via Automatic Test Case Generation of Programs. 10th Annual International Symposium on Applications and the Internet pp. 31 – 37 (2010)

[8] Shaukat Ali, Azhar Rauf, Huma Javed "SQLIPA:An authentication mechanism Against SQL Injection"

[9] Roichman, A., Gudes, E.: Fine-grained Access Control to WebDatabases. In: Proc. of 12th SACMAT Symposium, France (2007)

[10] K. Kemalis, and T. Tzouramanis (2008). SQL-IDS: A Specification-based Approach for SQLinjection Detection. SAC'08. Fortaleza, Ceará, Brazil, ACM: pp. 2153 2158.

[11] S. W. Boyd and A. D. Keromytis. SQLrand: Preventing SQL Injection Attacks. In Proceedings of the 2nd Applied Cryptography and Network Security Conference, pages 292–302, June 2004.

[12] 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

[13] G. Buehrer, B.W. Weide, P.A.G. Sivilotti, Using Parse Tree Validation to Prevent SQL Injection Attacks, in: 5th International Workshop on Software Engineering and Middleware, Lisbon, Portugal, 2005, pp. 106–113.

[14] P. Bisht, P. Madhusudan, and V. N. Venkatakrishnan. CANDID: Dynamic Candidate Evaluations for Automatic Prevention of SQL Injection Attacks. ACM Trans. Inf. Syst. Secur., 13(2):1–39, 2010

[15] Matthew Levine. The importance of application security. Technical report, @stake, jan 2003. http://www.atstake.com/research/reports/acrobat/atstake_application_security.pdf.

[16] The Open Web Application Security Project. A guide to building secure web applications, Version 1.1.1. Online Documentation, sep 2002. http://www.owasp.org/.

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

[18] Shubham Shrivastava, Rajeev Ranjan Kumar Tripathi, Attacks Due to SQL injection & their Prevention Method for Web-Application, International Journal of Computer Sciecne and information technologies, Vol 3 (2), pp.3615-3618, 2012.