Bulwark Against SQL Injection Attack– An Unified Approach

Prof (Dr.) Sushila Madan† and Ms Supriya Madan††

†Department of Computer Science, Lady Shri Ram College, University of Delh, India
†† Head of Department, Department of Information Technology, Vivekananda Institute of Professional Studies (Affiliated to GGSIP University), Delhi, India

Abstract
Data security has become a topic of primary discussion for security expert. Vulnerabilities are pervasive resulting in exposure of organizations and firms to a wide array of risks. Code Injection attack, a major concern for web security, occurs when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or when user input is not strongly typed and thereby unexpectedly executed, causing an error due to improper setup or coding such that the system fails to handle or properly respond to exceptional or unexpected data or conditions, which results in a situation wherein user credentials can be captured by injecting exceptional data. In spite of many tools and techniques, attacks on web application especially through SQL Injection Attacks are at a rise. Threat modeling is an important risk assessment and mitigation practice that provides the capability to secure a web application. A comprehensively designed threat model can provide a better understanding of the risks and help determine the extent of mitigation action. This paper aims to initiate the threat risk model ADMIRE which is a comprehensive, structured and stepwise approach, which would help to identify and mitigate Code Injections attacks and shield the database lying in the database servers, which may be unauthorizedly accessed for malafide reasons from the web applications.

***Keywords* :**
Security, SQL Injection, Threat modeling, Vulnerability, Web Application

1. Application Security

Web applications coupled with the communication technology provides an interface between the user and the database that results in user centric information which is very valuable and confidential. It is precisely for this reason, that web application security has become a primary topic of discussion for security experts, as attacks on application layer are constantly on rise. No matter how strong the firewall rule sets are or how diligent the patching mechanism may be, if the web application developers do not follow secured coding practices, attackers will gain easy unauthorized access to systems through port 80. One of the more common threats to web application is the SQL Injection vulnerability. SQL Injection derives from a software vulnerability, that allows a malicious user to inject custom code in the server engine by taking advantage of the unchecked assumptions the system makes about the inputs [1]. SQL Injection attacks are a top threat to today's internet [2]. With this kind of attacks, a malicious user can view sensitive information, destroy or modify protected data or even crash the entire application [3]. These attacks have proliferated in recent years causing severe security problems in systems and applications. Most web applications are typically developed in a General Purpose Language (GPL) for example PHP, ASP, along with a Domain Specific Language (DSL) which is used to address the needs of specific tasks. Hence language like SQL, XML play a very important role in the development of every modern web application. SQL Injection vulnerability is a type of security hole that is found in a multi-tiered application. An attacker can trick a database server into running an arbitrary, unauthorized, unintended SQL query by piggybacking extra SQL elements on top of an existing, predefined query that was intended to be executed by the application. The web application accepts user input and embeds this input inside an SQL query. This query is sent to the application's database server where it is executed. Also by providing certain malformed input, an attacker can manipulate the SQL query in such a way that its execution could have unintended consequences. To check if a web site is susceptible to SQL Injection Attack, techniques like (a) using input variations embedded in SQL query have number of enumerations like using ' to close string for example string='; password circumvention for example user=admin and password= ' or '1'='1; comment injection for example user=admin';-- and password='anything'; statement concatenation for example string='; INSERT INTO adminusers (username, password) VALUES ('hacked', 'xxx');--; (b) Another way to get almost any data from any database driven website application is by retrieving data from ODBC error messages. For example one can use information from error message produced by the MS SQL Server to get almost any data (c) Also the observations made on the use of system stored procedures would enable remote execution.
Almost all SQL databases and programming languages are potentially vulnerable. MS SQL Server, Oracle, MySQL, Postgres, DB2, MS Access, Sybase, Informix, accessed through applications developed using Perl and CGI scripts that access databases, ASP, JSP, PHP [4][5], XML, XSL, XSQL, MFC, and other ODBC-based tools and APIs. The

Manuscript received May 5, 2010
Manuscript revised May 20, 2010

more powerful the dialect of SQL supported by the database, the more susceptible the database is to attack [6]. The SQL Injection Attack can be broadly classified into two types. (a) Verbose: In this type, there is lack of error handling which provides a detailed feedback to the browser. It is highly prone to attack (b) Blind: In this type of attack the input is still vulnerable to SQL Injection, but error handling is performed to prevent ODBC errors from being displayed on the browser.
SQL injection attacks have been shown to be lethal. SQL injections tip copiousness of lists as a many prevalent equates to of attacking front-end Web applications and back-end databases to compromise data. In a "Breach Report for 2010" released by 7Safe progressing in February 2010, a whopping 60 percent of all breach incidents examined involved SQL injections [7].
According to Open Web Application Security Project report 2010 (OWASP) [8] Injection attacks are on the top. Similarly, new published reports research of a Web Hacking Incidents Database (WHID) shows SQL injections as a tip conflict vector, making up 19 percent of all confidence breaches examined by WHID [9]. According to Computer Security Institute [10] the leading reports on security, the menace of SQL injections has caused a major incursion into the area of cyber security and is one of the major contributors to financial frauds.

2. Common Logical Controls To Safeguard against SQL Injection Attacks

There are some common safe guards against SQL Injection attack which are simple to implement but due to ignorance and lackadaisical attitude regarding the security measures, precautions are not adhered to during designing and developing of web sites. SQL Injection attacks can be used like a sledgehammer or a scalpel and they are difficult to track down, hence security should be built into the Software Development Life Cycle to catch vulnerabilities as soon as possible. It has a known fact that the longer security vulnerability is left unsolved, it takes more efforts and are time consuming to resolve. The following safe guards are universally recommended.

1. Validation for all inputs is a must both at client side and server side: In security perspective, Server side validation is mandatory, client side validation can be easily bypassed by turning off javascript and vbscript in the browser. But for several other benefits, client side validation should be done along with server side.

2. Sanitize the input data : It is necessary to ensure that all valid data is accepted, while potentially dangerous data is rejected or sanitized. This can be difficult when valid characters or sequences of characters also have special meaning to the subsystem and may involve validating the data against a grammar For example, a simple approach to sanitising data that is displayed in a browser is to convert *“ to &quot, < to &lt , > to &gt .*

3. Use parameterized stored procedure with embedded parameters: This will prevent commands inserted from user input from being executed as the logic of a query is separated from its data. If one query is inadvertently bypassed, that could be enough to leave the application vulnerable. Example 1 shows a sample SQL statement that is SQL injectable.

Example 1 Query which is vulnerable to SQL Injection:

```
sSql = "SELECT LocationName FROM
Locations ";
sSql = sSql + " WHERE LocationID = " +
Request["LocationID"];
oCmd.CommandText = sSql;
```

The query shown in Example 2 utilizes parameterized queries, and is safe from SQL Injection attacks.

Example 2 Query not susceptible to SQL Injection:

```
sSql = "SELECT * FROM Locations ";
sSql = sSql + " WHERE LocationID =
@LocationID";
oCmd.CommandText = sSql;
oCmd.Parameters.Add("@LocationID",
Request["LocationID"]);
```

The application will send the SQL statement to the server without including the user’s input. Instead, a parameter- @LocationID- is used as a placeholder for that input. In this way, user input never becomes part of the command that SQL executes. Any input that an attacker inserts will be effectively negated. An error would still be generated, but it would be a simple datatype conversion error, and not something which an attacker could exploit.

4. Use a low privileged account to run the database : Running an application that connects to the database using the database's administrator account has the potential for an attacker to perform almost limitless commands with the database.

5. Delete system stored procedures In case the default installation of SQL Server is running as SYSTEM, the equivalent to Administrator Level in Windows then an attacker could use stored procedures like master..xp_cmdshell to perform remote execution (by intruding strings like ‘; exec master..xp_cmdshell .’-- to your SQL query). Therefore, delete stored procedures like

master..Xp_cmdshell, xp_startmail, xp_sendmail, sp_makewebtask . Example 3 is an excerpt from a USENET message, which reflects some common misconceptions about stored procedure

Example 3 showing misconceptions about stored procedure:

```
<%
strSQL = "sp_adduser '" &
Replace(Request.Form("username"),"'","'
'") & "','"
&
Replace(Request.Form("password"),"'","'
'") & "'," &
Replace(Request.Form("userlevel"),"'","
''")
%>
```

This was intended to show how a stored procedure can be safely called avoiding SQL Injection. The name 'sp_adduser' is intended to represent some user - written stored procedure rather than the system stored procedure of the same name. There are two observations made first, any query string that is composed with "''''" would imperil the security through SQL Injection attack even if it is calling a stored procedure. Second, closer examination of the final parameter will reveal that it is not delimited with single quotes. Presumably this reflects a numeric field, where the attacker to submit a 'userlevel' that looked like stated in Example 4

Example 4 :

```
1; shutdown --
```

...the SQL server would shut down, given appropriate privileges. Once you're submitting arbitrary SQL, single quotes are not needed because the 'char' function can be used to compose strings .

The Prepared Statement restricts the way the input can affect the execution of the statement. It is stated that batch SQL Statement can not be converted using *PreparedStatement*. The batch jobs cannot be handled since the JDBC Prepared Statement interface does not currently allow for multiple independent queries in the same batch Prepared Statement.

Many solutions have been put in literature for preventing SQL Injection Attacks. Another key preventive measure is to use bind variables in all programmatically generated SQL statements. One should not create a SQL string dynamically by concatenating the SQL statement with variable values. Bind variables prevent the use of quotes being injected into the SQL. A bind variable is a placeholder in a SQL command for a value that will be supplied at runtime by the application [25]. Bind variables can improve performance by eliminating the costly step of reparsing an application's SQL statements [20]. Bind variables are also valid in a LIKE clause and should be used. % and _ characters should be directly appended to the string as shown in Example 5 rather than concatenating the SQL statement as stated in Example 6 .

Example 5 SQL query using LIKE and % not susceptible to SQL Injection :

```
String name =
request.getParameter("name");
name =
query.append("%").append(name).append("
%");
 pstmt = conn.prepareStatement("SELECT
id FROM users WHERE name LIKE ?");
pstmt.setString (1, name);
```

Example 6 SQL query using Like and % susceptible to SQL Injection:

```
String name =
request.getParameter("name");
conn.prepareStatement("SELECT id FROM
users WHERE name LIKE '%" + name +
"%'");
```

Although Prepared Statements helps in defending against SQL Injection, there are possibilities of SQL Injection attacks through inappropriate usage of Prepared Statements To prevent SQL Injection a bind variable must be used with the PreparedStatement as shown in Example 7 and Example 8

Example 7 Bind variable with PreparedStatement :

```
String selectStatement = "SELECT * FROM
User WHERE userId = ? ";
PreparedStatement prepStmt =
con.prepareStatement(selectStatement);
prepStmt.setString(1, userId);
ResultSet rs = prepStmt.executeQuery();
```

Example 8 Bind variable with PreparedStatement

```
PreparedStatement pstmt =
conn.prepareStatement ("insert into EMP
(ENAME) values (?)");
String name =
request.getParameter("name");
pstmt.setString (1, name);
pstmt.execute();
pstmt.close();
```

The example below explains such a scenario where the input variables are passed directly into the Prepared Statement and thereby paving way for SQL Injection attacks as shown in Example 9.

Example 9 PreparedStatement vulnerable to SQL Injection :

```
String name =
request.getParameter("name");
PreparedStatement pstmt =
conn.prepareStatement("insert into EMP
(ENAME) values ('" + name + "')");
pstmt.execute();
pstmt.close();
```

<u>6. Error Handling</u> : Error messages are useful to an attacker because they give additional information about the database that might not otherwise be available. It is often thought of as being helpful for the application to return an error message to the user if something goes wrong so that if the problem persists they have some useful information to tell the technical support team. A better solution that does not compromise security would be to display a generic error message that simply states an error has occurred with a unique id. The unique id means nothing to the user, but it will be logged along with the actual error diagnostics on the server which the technical support team have access to.

3. Proposed Threat Risk Model ADMIRE for Countering SQL Injection Attacks

Security threat modeling is a process of assessing and documenting a system's security risk. Threat modeling enables one to understand a system's threat profile by examining it through the eyes of the potential foe. With techniques such as entry point identification, privilege boundaries and threat tree, one can identify strategies to mitigate potential threats to the system. The threat modeling efforts also enables the team to justify security features within a system, or security practices for using the system, to protect the corporate assets [11]. Threat modeling is the process of systematically deriving the key threats relevant to an application, in order to efficiently identify and mitigate potential security weakness before releasing it [12]. Threat modeling is a sound approach to addressing software risks at the design level [13]. Threat modeling is an essential process for secure web application development [14][15][16][17][18]. The need for the security necessitates the evaluation of the threat involved. As new technologies are developing, the security requirements and the measures to counter threats need to be constantly reviewed [19][20].

The authors of this paper have proposed the model ***ADMIRE*** a threat risk model which analyses risk assessment. ***ADMIRE*** as shown in *figure 1* is based on a structured and step wise approach for countering SQL Injection Attacks. The six steps enunciated are (i) ***A***nalyze the security objectives (ii) ***D***ivide the application (iii) ***M***ark the vulnerabilities (iv) ***I***dentify the threats (v) ***R***ank the threat (vi) ***E***liminate the threat.

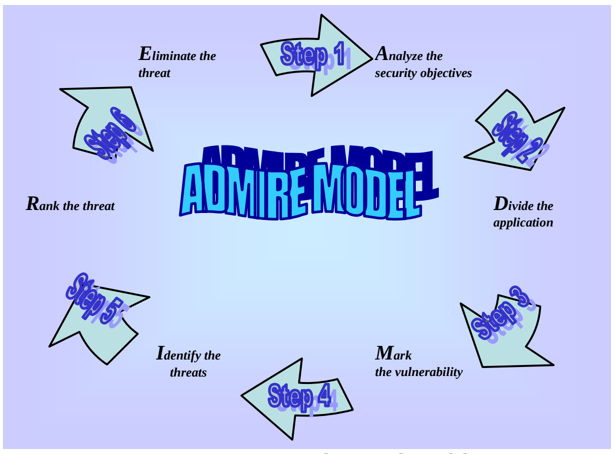


Figure 1: ADMIRE Threat Risk Model

The threat risk model – ***ADMIRE*** is explained in the following steps:

Step 1 - Analyze the security objectives

Identification of security objectives is the initial point of the threat model process and the various facets of security like identity abuse, financial risk, and loss of reputation, privacy intrusion, and protection of guarantees, regulations and corporate information security policy have to be the area of focus and goal. It is very vital to take into account the identification of the information as an asset that needs to be protected. In fact they represent the value the attacker is looking for. In this case the principal asset that we want to protect is the data stored in the database. Data protection involves satisfying two main requirements namely the integrity of the stored data and their confidentiality. In addition to data the second fundamental asset that needs to be protected is the data management service for which availability is crucial, the database should always be able to provide the data required by authorized users. It is very essential that organizations implement international security standard like ISO 27001, ISO 27002, OWASP, PCI/DSS, NIST to minimize the security failures. The increasing complexity of information intensive process has led to the development of security standards addressing the organizational aspect of IT security. These standards which codify industry best practice are used to drive the design and implementation of process. The compliance to a variety of information security standards is mandated by regulations and corporations need to act and implement a proactive information security enforcement strategy.

The interest of a security standard is to prevent the security failure and to mitigate their consequences. It has been widely recognized that information system security concerns the safeguarding of information system processes and data in any form. The technical control and measures, implements to achieve information security should be managerially monitored, reviewed and enhanced on a regular basis. With existing and emerging regulations in place, there are increasing uses of security standard by the industry to ensure that IT process and controls are adhering to the industrial best practices. Organizations have started the transition from being a network security centric organizations to a more application security centric organizations..

Step 2 - Divide the application

Divide the application by using an analysis method like data flow diagrams, business asset, data form, process and trust boundaries such as from the Internet to the web tier or from the business logic to the database server, need to be carefully analyzed. For example when investigating the authentication module, it is necessary to understand how data enters the module, how the module validates and processes the data, where the data flows and how the data is stored. It is necessary to identify the attack entry point of the system being analyzed. The entry point identified for the SQL injection attack is the normal web interface of the client in order to insert some malicious code or perform unauthorized or dangerous operation. It is very difficult to control this point because it is not possible to make any assumption about the client identity.

Step 3 - Mark the vulnerabilities

The next step is to mark the vulnerabilities. The vulnerabilities are like chinks in the armor which are exploited by the attackers. Categorizing these vulnerabilities is the most imperative step towards effective mitigation. To identify the Vulnerabilities the Microsoft Threat Model STRIDE which stands for ***S***poofing Identity, ***T***ampering with data, ***R***epudiation, **I**nformation disclosure, ***D***enial of service and ***E***levation of privileges has been used. STRIDE is useful to reduce attack surface area, improve design and eliminate vulnerabilities before they are released[21][22]. The acronym used to classify the different vulnerabilities by taking into account their effects on the security of the application. This model is helpful to developers and designers to identify and resolve security issues in the application code.

1. ***S****poofing* :- Spoofing Identity is when other illegal users access a computer and use the identity authentication such as the username and password.

Start with a single quote trick. Input something like: hi' or 1=1
Into login, or password, or even in the URL.
Example:
- Login: hi' or 1=1
- Pass: hi' or 1=1

2. ***T****ampering* :- Tampering is changing without authorization both the stored data and the data that is transmitted through open network.

Although tampering with SELECT ... WHERE statements can be very rewarding in many applications, attackers often want to be able to perform a UNION SELECT injection. Unlike WHERE clauses manipulation, successfully performing a UNION SELECT injection gives the attacker access to all tables in the system, which may not be otherwise accessible

3. ***R****epudiation*:- Repudiation is refusing some operation. When the system has no function of tracking illegal operation, some users would run illegal operations, resulting in having no proof to illustrate that an act was performed.

Attackers can have access to registry, can create a new administrator account remotely or change them through botnet without a track.

To change password for a specific login name
Eg EXEC sp_password ‘oldpass’, ‘newpass’,’username’

To show all the tables in the current database
Eg. EXEC sp_tables

4. ***I****nformation Disclosure*:- Information revealing includes information exposed to individuals who are authorized. For example, users can read the documents without granting access or intruder can read between the transmitting data in both computers.

SQL injection is exploiting the lack of input validation in an application (e.g. web site) to compromise the database/server behind it. Most programmers use the input provided by the user to build their SQL queries. If you accept input as valid and run it against a database system, it will be executed as it is. Example: user gives this input in a search field:

*SQL = "SELECT * FROM Products WHERE Name=''; drop table Orders--'"*

5. ***D**enial of Service*:- The denial of services occurs when an entity could not work well or its action prevents others from operating legal operations. The denial of services would lose the service to effective users.

The application is prone to an SQL-injection vulnerability because it fails to sufficiently sanitize input before using it in an SQL-query. The application is also prone to an unspecified denial-of-service vulnerability

An Attacker can remotely launch CMD_SHELL and can execute commands and even shutdown the whole server resulting in Denial of Service (DoS) attack

6. ***E**levation of Privilege*:- In privilege escalation, the users without the privileges get privileges.

*Use a low-privileged user that can only execute (certain) stored procedures, otherwise attacks could be possible through query like "SELECT * FROM sys.objects" to your server-side code, or launch extended procedures like xp_cmdshell, or drop object. The application user should not be sa or db_owner. Always lock down your applications, and only give the user the rights which are necessary.*

The table 1 below shows the mapping between the STRIDE categories and SQL Injection attacks

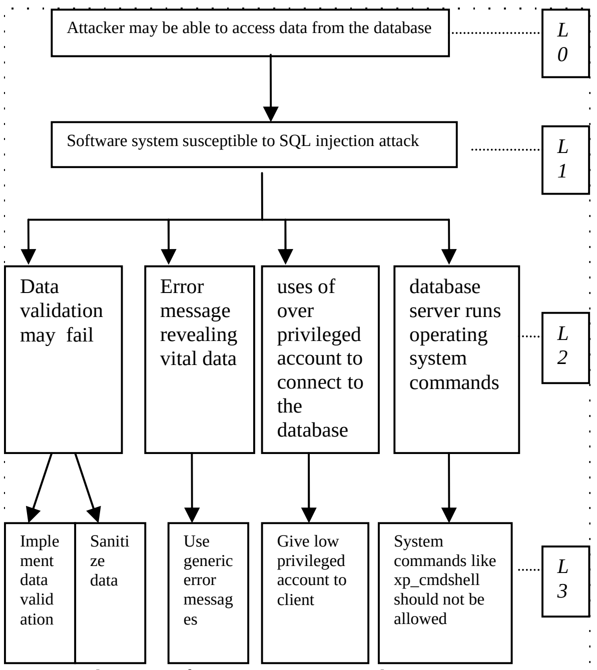
Table 1: Mapping STRIDE and SQL Injection

STRIDE	SQL Injection Attacks
Spoofing Identity	Possible
Tampering with data	Possible
Repudiation	Possible
Information disclosure	Possible
Denial of service	Possible
Elevation of privileges	Possible

Step 4 - Identify the threats

The next step is to identify the threats in correlation to their targets. Deploy one or more threat tree for each threat target. In order to identify threats Microsoft suggests two approaches for writing up threats. One is the threat graph and the other is a structured list. Security threats are modeled by attack trees, which describe the decision-making process attackers would go through to compromise the system. It is also vital to understand the level of attackers one is defending against.

The Figure 2 shows a threat tree to identify the SQL injection attack. A motivated attacker scrutinizes the web applications which have database connections and hence can be susceptible to SQL Injection attack (*L0 of Figure 2).* A SQL command with malicious intend is injected through the login page *(L1 of Figure 2).* The various threat entry points for SQL Injection attacks are scrutinized (*L2 of Figure 2*). The *L3 of Figure 2* depicts the security requirements and their objectives emanating from *L2*.


Attacker may be able to access data from the database
L 0
Software system susceptible to SQL injection attack
L 1
Data validation may fail
Error message revealing vital data
uses of over privileged account to connect to the database
database server runs operating system commands
L 2
Implement data validation
Sanitize data
Use generic error messages
Give low privileged account to client
System commands like xp_cmdshell should not be allowed
L 3
Figure 2: Threat tree for SQL Injection Attack

Step 5 - Rank the threat

Use a threat ranking method to determine the security risk for each threat tree. Once values are assigned to the different threats, order them for highest to lowest risk so as to enable to deal with them in a prioritized way. The threats must be rated on the basis of the risk exposure they present to the application architecture. A very popular rating system used for threat analysis is Microsoft's DREAD methodology. The acronym is formed from ***D**amage Potential, **R**eproducibility, **E**xploitability, **A**ffected Users and **D**iscoverability. DREAD can be used by reviewers to rank and enumerate threats in a structured way and produce similar risk ranking regardless of reviewers [23]. The DREAD algorithm shown below is used to compute the total risk value, which is an average of the risk ranking from the five categories.

Rank_DREAD = (Ranking from ***D***amage + Ranking from ***R***eproducibility + Ranking from ***E***xploitability + Ranking from ***A***ffected Users + Ranking from ***D***iscoverability Damage) / 5.
The calculation always produces a number between 0 and 10, the higher the number, the more serious the risk. Table 2 shows how SQL injection quantifies in the DREAD categories.

Table 2: SQL Injection attack quantified by DREAD

DREAD Categories	Description	Rank	Consequence of SQL Injection attack	SQL Injection Quantify
Damage Potential	If a threat exploit occur, how much damage will be caused?	0	Nothing	5 or 10
		5	Individual user data is compromised or affected	
		9	--	
		10	Complete system or data destruction	
Reproducibility	How easy is it to reproduce the threat exploit?	0	Very hard or impossible, even for administrators of the application	10
		5	One or two steps required, may need to be an authorized user	
		9	--	
		10	Just a web browser and the address bar is sufficient, without authentication	
Exploitability	What is needed to exploit this threat ?	0	Advance programming and networking knowledge with custom or advance attack tools	10
		5	Botnets on the internet or an exploit performed using available attack tools	
		9	--	
		10	Just a web browser	
Affected User	How many users will be affected ?	0	None	5 or 10
		5	Some user but not all	
		9		
		10	All user	
Discoverability	How easy is it to discover this threat?	0	Very hard, requires source code, administrative rights	9
		5	Can figure it out by guessing or monitoring network traces	
		9	Details of fault like this already in the public domain and can be easily discovered using search engine	
		10	The information is visible in the web	

Rigorous analysis of security requirements should be considered to detect security design flaws and corrective measures should be taken prior to costly development and deployment of flawed system. From Table 3 it is apparent that the risk derived from SQL Injection is ranked from high to very high. The business can prioritize actions on the basis of the assigned risk rating.

Table 3: Risk calculated using DREAD

	D	**R**	**E**	**A**	**D**	**Risk Avg**	**Remark**
Max. SQL Injection Risk quantify	10	10	10	10	9	(10+10 + 10+ 10+ 9) / 5 = **9.8**	Very High Ranking
Min. SQL Injection Risk quantify	5	10	10	5	9	(5+ 10+ 10+ 5+ 9) / 5 = **7.8**	High Ranking

The obvious benefit is the identification of previously unrecognized threats. But, even for previously considered threats, the mitigation action plans often do not correspond to the threat's risk level. The advantage of using the threat model is to apply or not to apply the appropriate mitigation corresponding to the risk level.

Step 6 - Eliminate the threat
After quantifying the risk, the team develops the risk response to the specific threat. Threats with low associated risk can be accepted. Moderate to high risk threats can be reduced through mitigation actions. Very high risk threats can be reduced through mitigation actions or avoided or options to transfer those risks might be sought. In order to eliminate risk it is necessary to ensure that the organization follows secured coding practices. It is also important that in order to mitigate SQL injection attack all preventive measures as stated in sub categories of L*3 of figure2* should be taken in to consideration. Hence the Defense in Depth Approach should be incorporated to ensure the web application is impregnable to SQL injection attacks. It is

further essential to repeat the sequential steps from step 3 of ***ADMIRE*** model in order to ensure that no further vulnerability exists.

Threat modeling is a continuous process. The threat scenario is dynamic. Continuous feedback to the threat modeling mechanisms from the systems of their interaction with the outside environment, latest technology updates and process reengineering are crucial for the successful implementation of a threat modeling system. Threat model ***ADMIRE*** will give the developers a better understanding of the application and help to discover bugs. It is a proactive security based analysis of an application and a crucial part of the design process.

4. Conclusion

Unless a web application has strong, centralized mechanism for validating all input from HTTP requests, vulnerabilities based on malicious inputs are very likely to exist. Securing computer systems should be a very important part of system design, development and deployment. The difficult part of building software is the specification, design and testing of this conceptual construct, not the implementation and testing of the implementation. Syntax errors will be made, but they are fuzz compared to the conceptual errors in most systems. Software flaws are caused because complexity makes software entities hard to design and to manage their development, and increasing complexity makes errors more probable. New software functions result in side effects that are difficult to predict, increasing the complexity rapidly as the software size grows. Software cannot be simplified by redesign because it has to conform with old programs which add to complexity. Software is subject to pressures for change all the time. Constant introduction of new features augment the pool of vulnerabilities.

A single unprotected query statement can result in compromising the security of the application, data or database server. Developers must be disciplined enough to apply the security methods to every web accessible procedure and function. Every dynamic query must be protected. It is apparent that safeguarding of security is becoming more difficult because the possible attack technologies are becoming increasingly sophisticated. Software rooted vulnerabilities like SQL Injections can be prevented, if the developers seriously incorporate the rule of validation while developing web applications. In spite the fact that the concept of validation is deep rooted and widely covered in almost all the International standards guidelines yet attacks are at a rise due to SQL Injection vulnerabilities. There is an urgent need to make the developers and users aware about the security standards and to encourage them to implement the standards meticulously, so as to minimize such attacks. It is also imperative to make these standards easily available, so that, their usage percolates down even to smaller organizations.

References

[1] Barrantes E, Ackley D, Forrest S, Palmer T, Stefanovic D and Zovi D, Randomized instruction set emulation to disrupt binary code injection attacks in CCS 2003", Proceedings of the 10th ACM Conference on Computer and Communication Security, pp281-289

[2] Xuxian Jiang; Wang, H.J.; Dongyan Xu; Yi-Min Wang; RandSys: Thwarting Code Injection Attacks with System Service Interface Randomization; Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on 10-12 Oct. 2007 Page(s):209 – 218

[3] Anley C, Advance SQL Injection in SQL Server Applications, Next Generation Security Software Ltd.

[4] Merlo, E.; Letarte, D.; Antoniol, G; Insider and Ousider Threat-Sensitive SQL Injection Vulnerability Analysis in PHP, IEEE WCRE 2006 13th Working Conference on Oct. 2006 Page(s):147 – 156

[5] Merlo, Ettore; Letarte, Dominic; Antoniol, Giuliano; Automated Protection of PHP Applications Against SQL-injection Attacks, IEEE CSMR 07 ; 11th European Conference on 21-23 March 2007 Page(s):191 – 202

[6] Expert Oracle Database Architecture: 9i and 10g Programming Techniques and Solutions (Apress, 2005)

[7] www.7safe.com/breach_report/Breach_report_2010.pdf

[8] www.owasp.org/index.asp

[9] http://www.acunetix.com/news/security-audit-results.htm

[10] CSI - www.gocsi.com/forms/csi_survey.html

[11] Frank Swiderski, Window Snyder," Threat Modeling (Microsoft Professional)", Microsoft Press 2004

[12] Howard and LeBlanc," Writing secure Code", Second edition

[13] Hoglund G and McGraw G , "Exploiting software: How to break code", Addison-Wesley Professional

[14] Zhimin Yang, Zenggung Zhang, "The Study on Resolution of STRIDE Threat Model", IEEE Conference

[15] Micheal Howard, James A.Whittaker, "Demystifying the Threat Modeling Process", IEEE Computer Society

[16] Xiaohong Li, Ke He, "A Unified Threat Model For Assessing Threat in Web Applications, 2008 International Conference on Security and Assurance, IEEE

[17] Linzhang Wang, Eric Wong," The Threat Model Driven Approach for Security Testing",Third Internation Workshop on Software Engineering for Secure System (SESS'07"),IEEE

[18] Jesper M.Johansson, "Network Threat Modeling", Twelfth IEEE International Workshops on Enabling Technologies (WETICE '03)

[19] Nazir A. Malik , Muhammad Younus Javad, Umar Mahmud, "Threat Modeling in Pervasive Computing Paradigm",ESR Groups France, IEEE

[20] Bruni Romero, Marianella Villegas, Marina Meze, "Simon's Intelligence Phase for Security Risk Assessment in Web Application", Fifth International Conference on Information Technology:New Generation, IEEE

[21] Visveswaran Chidambaram, "Threat Modeling in Enterprise Architecture Integration", EA & BC Vil 2 No 4, December 04

[22] Threat Modeling, Pattern and Practices, MSDN Microsoft Corporation
[23] Threats and Counter measures, Web Security Threats and Countermeasures, Patterns and Practices MSDN, Microsoft Corporation.