Safe Guard Anomalies against SQL Injection Attacks

Romil Rawat
M.Tech Scholar
Deptt. Of IT, SATI Vidisha
M.P., India

Chandrapal Singh Dangi
M.Tech Scholar
Deptt. Of CSE, SSSIST
Sehore, M.P., India

Jagdish Patil
M.Tech Scholar
Deptt. Of IT, SATI Vidisha
M.P., India

ABSTRACT
For internet, web application exists and for web application syntax, semantics, coding and design exists, and for coding and designing, algorithm exists, and for algorithm, protecting techniques and rules exists, But as the internet technologies advanced, vulnerability also advanced .Various old procedures, algorithm functions, coding and designing syntax and semantics are there, which are vulnerable to attack and if used could be easily traced or hacked by the attacker. Old practices which are vulnerable should be banned in organization, companies and govt. sectors and secure guidelines should be issued, which consists of security guildelines.and should be strictly followed. In this paper we have proposed coding flaws at different platforms and their solutions.

Keywords
SQL Injection, Database Security, Authentication, HTTP

1. INTRODUCTION

With the increase use of web application, there is a chase of getting more and more projects, designs and new standards [1, 2, 3]. Various techniques and various methods are there for creating web application and various languages exists for its creation. Thousands of Companies are there and lakes of developers are also there, for doing internet related work for that company. And they are all not following global standard for writing language code and designing framework. And, [4, 5] as a result some loopholes are left created in their application, which result in vulnerable cause for that application. If proper Input validation, syntax validation, secure coding framework, security guidelines are to be followed, the application become secure. Code reviews are an excellent means of checking for common mistakes to enforce good security coding practices. Coding standard can guide developers to make secure standard design practices. The inexperienced programmer often builds the SQL statement as a string, appending the input data into the string along the way [6, 7]. The programmer then submits the entire string to the database as a statement, allowing malicious input to be interpreted directly by the database engine. Use of prepared statement makes a SQL statement secure because it binds user input to those placeholders,anathor techniques is to use input filtering[7,8] techniques to filter user input meta characters, many languages provide inbuilt function and packages to filter input parameters. Various tools are also available which checks web application for different vulnerability points. Normally, web applications is a three tier architecture, the Application tier at the user side, Middle tier which converts the user queries into the SQL format, and the backend database server which stores the user data as well as the user's authentication table. Whenever a user wants to enter into the web database through application tier, the user inputs his/her authentication information from a login interface.

Working principal of web application is as follows:-

- Web application is requested through a web browser by a user.
- The HTTP protocol accepts a request of user and sent to the targeted web server.
- Server executes the request received
- Application program generates a output and sent back to the user via HTTP.
- Current states of User, Web server and their execution report are maintained by a special unit called cookies.

In our paper we have presented coding standards, and secure guideline practices for developer and designers.

2. RELATED WORK

The mechanism to keep track of the positive taints and negative taints is proposed by William G.J. Halfond, Alessandro Orso, Panagiotis Manolios [18],

Defensive Programming [11][12] has given a approach for Programmers by which they can implement their own input filters or use existing safe API s that prevent malicious input or that convert malicious input in to safer input. . Vulnerability pattern approach is used by Livshits and Lam [16], they propose static analysis approach for finding the SQL injection attack. . The main issues of this method, is that it cannot detect the SQL injection attacks patterns that are not known beforehand. Vulnerability patterns are described here in this approach.

AMNESIA mechanism to prevent SQL injection at run time is proposed by William G.J. Halfond and Alessandro Orso [17].It uses a model based approach to detect illegal queries before it sends for execution to database

Static analysis framework (called SAFELI) has been proposed by Xiang Fu et al [13], for identifying SIA (SQL Injection attacks) vulnerabilities at compile time.. the source code information can be analyzed by SAFELI and will be able to identify very delicate vulnerabilities that cannot be discovered by black-box vulnerability scanners.

Scott and Sharp Proxy filter [9] [10] , this technique can be effective against SQLIA; they used a proxy to filter input

data and output data streams for a web application ,although correctly specify filtering rules for each application is required by the developers to input.

The mechanism which filters the SQL Injection in a static manner is proposed by Buehrer et al [15]. The SQL statements by comparing the parse tree of a SQL statement before and after input and only allowing to SQL statements to execute if the parse trees match.

Novel-specification based methodology proposed by Konstantinos et al [14], they given a mechanism to detect SQL injection with. This approach utilizes specifications that define the intended syntactic structure of SQL queries that are produced and excecuted by the web-application.

Instruction–Set Randomization [9][11] defined a framework that allows developers to create SQL queries using randomized keywords instead of the normal SQL keywords.

Marco Cova et al [19], proposed a Swaddler which, analyzes the internal state of a web application and learns the relationships between the application's critical execution points and the application's internal state..

3. GENERAL GUIDELINE

• Use of good software design engineering and other phases of software facilitates a structured, small, and simple code. Always follow secure coding checklist.

• Use of Secure libraries which contains standard designs (e.g., anti-cross site scripting library) to protect against security bugs during web application development. Application scanner should be used to test code to detect vulnerabilities.

Reading security-related forums, magazines, research papers, and newsletters. And to apply these researches on design and implementation work.

• Use of the latest compilers also recommends defenses against coding errors; for example, GCC protects code from buffer overflows.

• Proper error/exception handling is to be followed. Check the return values of every function, especially security-related functions.

• Use of limited permission to access database, always uses SQL execute-only permission.

Because it limits uses of given accesses, securing the design, don't use high-privileged accounts like sysadmin or dbo. If it is needed to use high-privileged operations, then wrap those operations in a stored procedure and stored procedure should be signed with a certificate that has the required high privileges and grant execute permission on the stored procedure.

• Always follow input validation. Create 'white list' of good input parameters and 'Black list ' of attacking and bad parameters.

•Use version/configuration control to track changes occurred in the code or document. This will create compatibility wizards for moving across versions.

• Use parameterized SQL statement. And SQL Statement could be vulnerable, so never trust on input for SQL statement.

• The function, syntaxes and designs which seems to be vulnerable and creates weak application should be banned strictly; rules should be created to follows only secure and safe guidelines.

• For keeping client web browser secure from interpreting by malicious activity, Encode HTML input.XSS attacks cause serious problem if proper encoding is not used.

• Don't use sensitive data in cookies; it could create cookie theft attack.

•java.util.regex, SqlParametersCollection, PHP Data Object (PDO) and Perl's DBI library. Use these libraries, package and function to escape any potentially dangerous characters.

•Use strong cryptographic techniques for encrypt all confidential data. Use secure algorithms with long keys.

• Developers should be guided or educated by the modern attack scenario.traing and workshop programs should be incorporated at regular interval by experts.

4. FOLLOWS THE SECURE CODING GUIDELINES

These languages are frequently used for web development, and they support various predefined packages, function, design and parameters. If they are not properly sanitized and quoted, it could result disastrous result. The main security issue in web designing is keeping database confidential and accessible to unauthorized user. If SQL query generated by user input is properly handle before creation of dynamic SQL, the application will work in safe mode, but if any designing and parameter transfer flaws occurred due to Input parameter, it will create unauthorized access and bypassing of web application by vulnerability intension. Below is some codes and their designing solution, which will create safe Code.

4.1 Secure coding with VB.NET

4.1.1 Vulnerable piece of .Net code

```
dim rolno As String
‘ Get name from user input
dim name As String = Request.QueryString("name")
‘ Setup the database connection (details omitted)
Dim connection As SqlConnection = New
SqlConnection(<setup db conn here>)
‘ Substitute the unfiltered “name” parameter in the query string
Dim query As String = String.Format( _
"SELECT rolno FROM student  WHERE Name='{0}'", _ name)
‘ Send command to database
Dim query As SqlCommand = New SqlCommand(queryText,
connection)
Dim dataAdapter As SqlDataAdapter = New
SqlDataAdapter(query)
GetRolnoFromName = New DataSet
dataAdapter.Fill(GetRolnoFromName)
```

4.1.2 Secure Code of .Net code

```
dim rolno  As String
‘ Get name from user input
dim name As String = Request.QueryString("name")
‘ Setup the database connection (details omitted)
Dim connection As SqlConnection = New
SqlConnection(<setup db conn here>)
‘ Set up the query to use @name where we will add that
parameter via
‘ the SqlParametersCollection
Dim queryText As String = String.Format( _ "SELECT rolno
FROM student WHERE name=@name")
‘ Send command to database
Command(queryText, connection)
Dim query As SqlCommand = New
Sqlquery.Parameters.Add(@name,name)
Dim dataAdapter As SqlDataAdapter = New
SqlDataAdapter(cmd)
GetRolnoFromName = New DataSet
dataAdapter.Fill(GetRolnoFromName)
```

4.2 Secure coding with Perl

4.2.1 Vulnerable Perl piece of Perl code:

```
$query = $sql->prepare(“select rolno from student  where name
= ‘$name’“);
$query->execute;
```

4.2.2 Secure Code

```
$query = $sql->prepare(“select rolno  from student  where name
= ?“);
$query->execute($name);
```

4.3 Secure coding with PHP

4.3.1 Vulnerable piece of PHP code

```
// Get $name from the HTTP request...
$name = $_REQUEST(“name”);
// Set up our query to look up ROLNO by name...
$query = “select rolno from student  where name = ‘” . $name .
“’”;
        // Now, execute our query...
$query->execute();
// Now, pull our rolno out of the query results...
$rolno->$query->fetchAll();
```

4.3.2 Secure Code

```
// Get $name from the HTTP request...
$name = $_REQUEST(“name”);
// Build our query to lookup ROLNO by name,
// using a prepared statement and substitution to bind our
parameter...
$query = $dbhandle->prepare(“select rolno  from student  where
name = ?”);
// Now execute the query, substituting the $name variable using
the
// an array to satisfy the API.
$query->execute($query,array($name));
// Now, pull our ROLNO out of the query results...
        $rolno = $query->fetchAll();
```

4.4 Secure coding with Java

4.4.1 Vulnerable piece of java code

```
// Build our query, directly substituting the NAME parameter
from input
String query = “select rolno  from student  where name = ‘
+ req.getParameter(“NAME”) + “’”
// Get a Statement object from the Connection object
Statement statement = connection.createStatement();
// Execute our query, allowing the attacker to supply his SQL
commands...
ResultSet results = statement.executeQuery(query);
// Get the ROLNO from the result set..
```

4.4.2 Secure Code

```
// Build Prepared statement, passing our query string in as a
parameter...
PreparedStatement statement = connection.prepareStatement
("SELECT rolno FROM  student WHERE name = ?“);
// Substitute that “?” symbol for our the NAME, noting that it’s
to
// be bound as a String parameter...
statement.setString(1,req.getParameter(“NAME”);
// Execute query and get query results...
ResultSet results = statement.executeQuery(query);
// Get the ROLNO  from the result set.
String rolno = results.getString(1);
```

5. EVALUATION

Web Application is compared before and after applying the secure guideline principles. And it is found that if checkpoints and secure methods are used, the performance and security of application increases to much extent and it becomes almost impossible to bypass the web application by unauthenticated manner. As shown in both graphs the attack intensity reduces to much extent after applying the secure guidelines. Design flaws are removed by checkpoints at every vulnerable position. Figure 1 shows the increase of attack intensity, if design flaws and lack of secure guidelines are there. Figure 2 shows the decrement in attack intensity and increases the security. Performance of application has been increased.

6. CONCLUSION

Our concept provides a secure application, based on secure guidelines. Here application loopholes are closed by secure checkpoints. Web application is vulnerable only, if there is designing flaws, coding mistakes and lack of proper guidelines. Our system used advanced guidelines, which shows some rules which should be strictly followed by designer and coder. Some old methods which are vulnerable to attack should be strictly banned. Expert ways are used to eliminate any possible attack scenario and its terror.

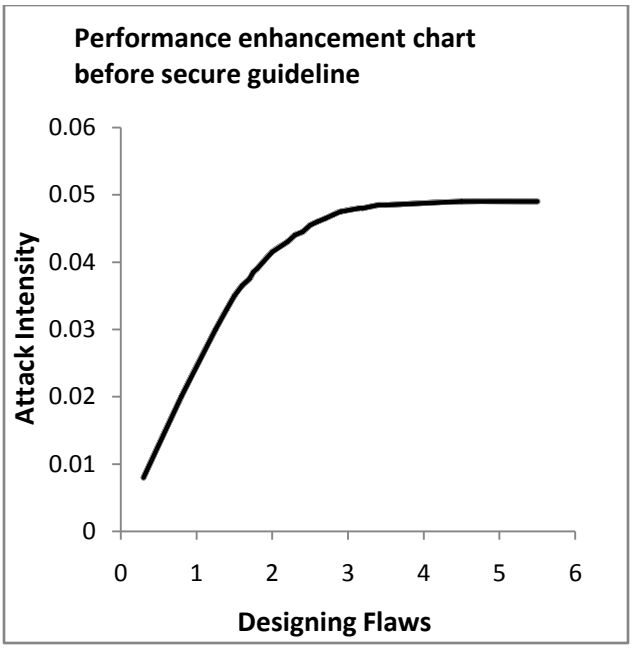

Fig.1. Increase of Attack Intensity

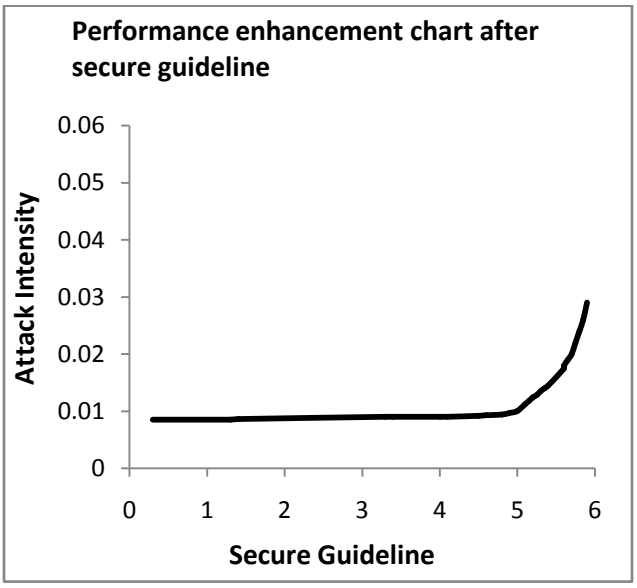

Fig.2. Decrement in Attack Intensity

7. REFERENCES

[1] Monica S. Lam, Michael Martin, Benjamin Livshits, and John Whaley, "Securing Web Applications with Static and Dynamic Information Flow Tracking," PEPM'08, January 7–8, 2008, San Francisco, California, USA.

[2] J. Scambray, M. Shema, and C. Sima, Hacking Exposed Web Applications, 2nd ed., McGraw-Hill, 2006.

[3] D. Stuttard and M. Pinto, The Web Application Hacker's Handbook,Wiley Publishing, 2008.

[4] Michael Howard, David LeBlanc, "Writing secure code", Microsoft Press, 2003.

[5] Common Weaknesses Enumeration Definitions, April-2010, http://cwe.mitre.org/data/definitions/113.html.

[6] Mark G. Graff, Kenneth R.van Wyk, 'Secure Coding Principles, and Practices', 2003.

[7] Trupti Shiralkar and Brenda Grove "Guidelines for Secure Coding", January,2009.

[8] B. Indrani & E. Ramaraj ," X – LOG AUTHENTICATION TECHNIQUE TO PREVENT SQL INJECTION ATTACKS", International Journal of Information Technology and Knowledge Management January-June 2011, Volume 4, No. 1, pp. 323-328.

[9] A Classification of SQL Inject ion At tacks and Countermeasures: William G.J. Hal Fond and Alessandro Orso, Col lege of Comput ing, Georgia Institute of Technology.Gatech.edu.

[10] D. Scott and R. Sharp, "Abstracting Application-level Web Security", In Proceedings of the 11th International Conference on the World Wide Web (WWW 2002), Pages 396–407, 2002.Y. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. Lee, and S. Y. Kuo.

[11] "Securing Web Application Code by Static Analysis and Runtime Protection", In Proceedings of the 12th International World Wide Web Conference (WWW 04), May 2004.

[12] SQL Injection Attack Examples based on the Taxonomy of Orso et al.

[13] Xiang Fu, Xin Lu, Boris Peltsverger, Shijun Chen, "A Static Analysis Framework For Detecting SQL Injection Vulnerabilities", IEEE Transaction of computer software and application conference, 2007.

[14] Konstantinos Kemalis and Theodoros Tzouramanis, "Specification based approach on SQL Injection detection", ACM, 2008.

[15] G.T. Buehrer, B.W.Weide and P.A..G.Sivilotti, "Using Parse tree validation to prevent SQL Injection attacks", In proc. Of the 5th International Workshop on Software Engineering and Middleware(SEM '056), Pages 106-113, Sep. 2005.

[16] V.B. Livshits and M.S. Lam, "Finding Security vulnerability in java applications with static analysis", In proceedings of the 14th Usenix Security Symposium, Aug 2005.

[17] William G.J. Halfond, Alessandro Orso,Panagiotis Manolios, "WASP:Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation", IEEE Transaction of Software Engineering Vol 34, Nol, January/February 2008.

[18] W.G. J. Halfond and A. Orso, "Combining Static Analysis and Run time monitoring to counter SQL Injection attacks", 3rd International workshop on Dynamic Analysis, St. Louis, Missouri, 2005, pp.1.

[19] Marco Cova, Davide Balzarotti, Viktoria Felmetsger, and Giovanni vigna, " Swaddler: An approach for the anamoly based character distribution models in the detection of SQL Injection attacks", Recent Advances in Intrusion Detection System, Pages 63-86, Springerlink, 2007.