RESEARCH ARTICLE OPEN ACCESS

Hybridizing Regular Expression with query processing to remove SQL Injection and XSS attacks

Monali Sachin Kawalkar[1], Dr. P. K. Butey[2]
[1]*Department of computer science R.T.M.Nagpur University*
[2]*Head of the Department Department of Computer Science Kamla Nehru, R.T.M Nagpur University*

ABSTRACT
Web applications are widely used everywhere like e-commerce, online payments, online banking, money transfer, social networking, etc. As web application interacts with database where critical information is stored over the network. The methodology used is Structure Query language (SQL) and Scripting language. OWASP [2] has released the latest version of "Top 10 Vulnerabilities" based on the previous incidents as well as on the risks associated with the Vulnerabilities. SQL Injection and Cross Site Scripting are the most serious security threat to Web applications they allow attackers to obtain unrestricted access to the databases underlying the applications and to the potentially sensitive information these databases contain. Cross Site Scripting is a most prevalent web application security issue. This occurs when application sends the user provided data to the web browser without validating or encoding the account. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites. To address this problem, we describe review of the different types of SQL injection attacks and XSS attack. For each type of attack, we provide descriptions and examples of how attacks of that type could be performed. This paper identifies the security of a full blown E-Commerce website, and checks for any SQL Injection or XSS vulnerabilities based on an hybrid approach of query processing and regular expression checking. This approach improves the accuracy of attack detection and sanitizes the input query, so that the back end server is secured. In our case, we tested the system over more than 50 different types of SQL Injection and XSS attacks, and found it 100% accurate in terms of attack detection and query sanitization
Keywords- OWASP – Open Web Application Security Project (OWASP) , SQL – Structure Query language, Web Application ,Detection and Prevention Techniques. XSS – Cross site scripting, Input query sanitization, DOM - Document Object Model.

Date of Submission: 16-11-2017 Date of acceptance: 30-11-2017

I. INTRODUCTION

Web Applications are vulnerable to a variety of new security threats getting generated everyday by various sources, these applications which are hosted on Internet which is a widespread information infrastructure. Unaware of the Confidentiality, Integrity, Availability, Security and Privacy, the internet is becoming a repository of Business critical information. No matter which business, Information and data of organization is the most important business asset in today's environment, this can be achieved an appropriate level of Information security.

Websites rely heavily on complex web applications to deliver different output or content to a wide variety of users according to set preferences and specific needs. This arms organizations with the ability to provide better value to their customers and prospects. However, dynamic websites suffer from serious vulnerabilities rendering organizations helpless and committed to Sql injection attacks and cross site scripting attacks on their data. E-commerce sites are tricked by attackers and they lead into shipping goods for no charge, usernames and passwords have been cracked, and confidential and important credentials of users have been leaked. if there is no validation on the input of the application, then the malicious code can steal sessions, cookies, or inject and show private data for the user

SQL injection and Xss vulnerabilities has been described the most serious threats for Web applications [33].Web applications that are vulnerable to SQL injection may allow an attacker to gain complete access to their underlying databases. Because these databases often contain sensitive user information, the resulting security violations can include identity theft, loss of confidential information, and fraud. In some cases, attackers can even use an SQL injection vulnerability to take control of and corrupt the system that hosts the Web application.

These attacks are a serious threat to any Web application that receives input from users and incorporates it into SQL queries to an underlying database. Most Web applications used on the Internet or within enterprise systems work this way and could therefore be vulnerable to SQL injection. Attacks can be characterized as Identifying injectable parameters, Performing database finger-printing, Determining database schema, Extracting data, Adding or modifying data, Performing denial of service, Evading detection, Bypassing authentication, Executing remote commands, Performing privilege escalation.

Example Application

This section shows an example application that contains SQL injection vulnerability. We use this example in following section to provide attack examples.

```
String strUserId="";
String strUserpassword ="";
String strUserName="";
String strEmail="";
strUerId = request.getParameter("user_id");
strUserpassword = request.getParameter("user_pwd");
Connection = DriverManager.getConnection("lead");
String ssql = " select user_email from user_mst where user_id="'+ strUerId +'" and user_pwd="'+ strUserpassword +"' ";
Statement st1 = connection.createStatement();
ResultSet rs1 = st1.executeQuery(ssql);
If (rs1!=NULL)
        displayEmail(rs1);
Else
        displayAuthFailed();
```

Figure 1.1 Extract of servlet implementation

This example refers to a quite simple vulnerability that could be prevented using a basic coding attach. We use this example simply for illustrative purposes because it is easy to understand and general enough to illustrate many different types of attacks. The code extract in Figure 1.1 implements the login functionality for an application. It is based on similar implementations of login functionality that we have found in existing Web-based applications. The code in the example uses the input parameters user_id, user_pwd, and user_name to dynamically build an SQL query and submit it to a database. For example, if a user submits user_id, user_pwd, and user_name as "`0001,`" "`secret,`" and "`Smith`," the application dynamically builds and submits the query:

The cause of SQL injection vulnerabilities is mainly insufficient validation of user input. To address this problem, developers have used a range of coding guidelines such as OWASP testing guidelines [3,4] that promote defensive coding practices, such as encoding user input and validation.

```
SELECT user_email from user_mst where user_id='0001' and user_pwd='secrete' and user_name='Smith'
```

If the User Id, User password, and User name match the corresponding entry in the database, `Smith's` account information is returned and then displayed by function `displayEmail()`. If there is no match in the database, function `displayAuthFailed()` displays an appropriate error message.

SQL injection attack types and XSS attack types

In this section we describe the different kinds of SQLIA[34]. For each attack type we provide an attack example. The different types of attacks are generally not performed in isolation; many of them are used together or sequentially, depending on the specific goals of the attacker. Similarly describes xss attack types.

Tautologies

In this type of attack the attacker goal is bypassing authentication, identifying injectable parameters, extracting data. The general goal of a tautology-based attack is to inject code in one or more conditional statements so that they always evaluate to true. The most common usages are to bypass authentication pages and extract data. In this type of injection, an attacker exploits an injectable field that is used in a query's WHERE conditional. Typically, the attack is successful when the code either displays all of the returned records or performs some action if at least one record is returned.

Example: In this example attack, an attacker submits " ' or 1=1 - -" for the login input field (the input submitted for the other fields is irrelevant). The resulting query is:

```
SELECT email FROM user_mst WHERE User_id='' or 1=1 -- AND user_pwd='' AND user_name=
```

The code injected in the conditional (OR 1=1) transforms the entire WHERE clause into a tautology. The database uses the conditional as the basis for evaluating each row and deciding which ones to return to the application. Because the conditional is a tautology, the query evaluates to true for each row in the table and returns all of

them. In our example, the returned set evaluates to a non null value, which causes the application to conclude that the user authentication was successful. Therefore, the application would invoke method `displayEmail()` and show all of the emails in the set return by the database.

Illegal/Logically Incorrect Queries

In this type of attack attacker Identifying injectable parameters, performing database finger-printing, extracting data. Attacker collects important information about the type and structure of the back-end database of a Web application. The attack is considered a preliminary, information gathering step for other attacks. If the attack is successfully executed the default error page returned by application servers. When performing this attack, an attacker tries to inject statements that cause a syntax, type conversion, or logical error into the database. Logical errors often reveal the names of the tables and columns that caused the error.

Example: This example attack's goal is to cause a type conversion error that can reveal relevant data. To do this, the attacker injects the following text into input field user_id: "convert(int,(select top 1 name from sysobjects where xtype='u'))". The resulting query is:

```
SELECT email FROM user_mst WHERE
user_name='' AND
User_pwd='' AND user_id= convert
(int,(select top 1 name from
sysobjects where xtype='u'))
```

In the attack string, the injected select query attempts to extract the first user_mst table (xtype='u') from the database's metadata table (assume the application is using Microsoft SQL Server, for which the metadata table is called sysobjects). The query then tries to convert this table name into an integer. Because this is not a legal type conversion, the database throws an error. For Microsoft SQL Server, the error would be: *"Microsoft OLE DB Provider for SQL Server (0x80040E07) Error converting nvarchar value 'user_mobile'to a column of data type int."*

There are two useful pieces of information in this message that help an attacker. First, the attacker can see that the database is an SQL Server database, as the error message explicitly states this infomation. Second, the error message reveals the value of the string that caused the type conversion to occur. In this case, this value is also the name of the first user-defined table in the database: "user_mst." A similar strategy can be used to systematically extract the name and type of each column in the database. Using this information about the schema of the database, an attacker can then create further attacks that target specific pieces of information.

Union Query

In this type of attack attacker bypassing authentication and extracting data. In union-query attacks, an attacker exploits a vulnerable parameter to change the data set returned for a given query. With this technique, an attacker can trick the application into returning data from a table different from the one that was intended by the developer. Attackers do this by injecting a statement of the form: UNION SELECT <rest of injected query>. Because the attackers completely control the second/injected query, they can use that query to retrieve information from a specified table. The result of this attack is that the database returns a dataset that is the union of the results of the original first query and the results of the injected second query.

Example: An attacker could inject the text "' UNION SELECT tel_cell from trncustomer_new where
Customer_id=110005 - -" into the login field, which produces the following query:

```
SELECT email FROM users WHERE
user_name='' UNION
SELECT tel_cell from
trncustomer_new where
Customer_id=110005 -- AND
user_pwd='' AND user_id=
```

Assuming that there is no login equal to "", the original first query returns the null set, whereas the second query returns data from the "`trncustomer_new`" table. In this case, the database would return column "tel_cell" for account "`110005`." The database takes the results of these two queries, unions them, and returns them to the application. The effect of this application is that the value for tel_cell is displayed along with the customer information.

Piggy-Backed Queries

Attacker goal is extracting data, adding or modifying data, performing denial of service, executing remote commands. In this attack type, an attacker tries to inject additional queries into the original query i.e they are trying to include new and distinct queries that "piggy-back" on the original query. As a result, the database receives multiple SQL queries. The first is the intended query which is executed as normal; the subsequent ones are the injected queries, which are executed in addition to the first. Vulnerability to this type of

attack is often dependent on having a database configuration that allows multiple statements to be contained in a single string.

Example: If the attacker inputs "'; drop table users --" into the passfield, the application generates the query:

```
SELECT email FROM user_mst WHERE
user_name='Smith' AND
User_pwd=''; drop table users -- '
AND user_id=0001
```

After completing the first query, the database would recognize the query delimiter (";") and execute the injected second query. The result of executing the second query would be to drop table users, which would likely destroy valuable information.

Stored Procedures

In this type of attack, an attacker performing privilege escalation, performing denial of service and executing remote commands. SQLIAs of this type try to execute stored procedures present in the database .Most of the time developer uses databases with a standard set of stored procedures that extend the functionality of the database and allow for interaction with the operating system. Therefore, once an attacker determines which backend database is in use, SQLIAs can be crafted to execute stored procedures provided by that specific database, including procedures that interact with the operating system. In a common way that writing a stored procedures for web applications are invulnerable to SQLAs i.e stored procedure may contain other type of vulnerabilities such as buffer overflows, that allow attacker to run arbitrary code on the server or escalate their privileges.

```
CREATE PROCEDURE
DBO.isAuthenticated
@user_name varchar2, @user_pwd
varchar2, @user_id int
AS
EXEC("SELECT email FROM user_mst
WHERE user_name='" +@user_name+ "'
and user_pwd='" +@user_pwd+
"' and user_id=" +@user_id);
GO
```

Figure 1.2 Stored procedure for checking identification.

Example: This example describes a arameterized stored procedure can be exploited via an SQLA. In the above example, suppose we assume that the query string constructed at line 8 and 9 has been replaced by a call to the stored procedure defined in Figure 1.2. The stored procedure returns a true/false value to indicate whether the user's credentials authenticated correctly. To execute an SQLIA, the attacker simply injects " ' ; SHUTDOWN; --" into either the user_name or user_pwd fields. This injection causes the stored procedure to generate the following query:

```
SELECT email FROM user_mst WHERE
User_name='Smith' AND user_pwd='
'; SHUTDOWN; -- AND user_id=
```

At this point, this attack works like a piggy-back attack. The first query is executed normally, and then the second, malicious query is executed, which results in a database shut down.

Inference

In this type of attack attacker identifying injectable parameters, extracting data and determining database schema. In this attack, the query is modified to recast it in the form of an action that is executed based on the answer to a true/-false question about data values in the database. In this type of injection, attackers are generally trying to attack a site that has been secured, when an injection has succeeded, there is no usable feedback via database error messages. Inference attack includes two techniques first is blind injection and second is timing attack.

Blind Injection- In this technique, the information must be inferred from the behavior of the page by asking the server true/-false questions. If the injected statement evaluates to true, the site continues to function normally. If the statement evaluates to false, although there is no descriptive error message, the page differs significantly from the normally-functioning page.

Timing Attack- A timing attack allows an attacker to gain information from a database by observing timing delays in the response of the database. To perform a timing attack, attackers structure their injected query in the form of an if/then statement e.g. the WAITFOR keyword, which causes the database to delay its response by a specified time.

Example: In two ways Inference based attacks can be used. The first of these is identifying injectable parameters using blind injection. Consider two possible injections into the login field. The first being "legalUser' and 1=0 - -" and the second, "legalUser' and 1=1 - -". These injections result in the following two queries:

```
SELECT email FROM user_mst WHERE
user_name='legalUser'
and 1=0 -- ' AND user_pwd='' AND
user_id=0
```

```
SELECT email FROM user_mst WHERE
user_name='legalUser'
and 1=1 -- ' AND user_pwd='' AND
user_id=0
```

If we considering two scenarios, in the first scenario, we have a secure application, and the input for user_name is validated correctly. In this case, both injections would return login error messages, and

the attacker would know that the user_name parameter is not vulnerable. In the second scenario, we have an insecure application and the user_name parameter is vulnerable to injection. The attacker submits the first injection and, because it always evaluates to false, the application returns a login error message. At this point however, the attacker does not know if this is because the application validated the input correctly and blocked the attack attempt or because the attack itself caused the login error. The attacker then submits the second query, which always evaluates to true. If in this case there is no login error message, then the attacker knows that the attack went through and that the login parameter is vulnerable to injection.

The second way inference based attacks can be used is to perform data extraction. Timing based inference attack to extract a table name from the database. In this attack, the following is injected into the *user_name* parameter:

```
''legalUser' and
ASCII(SUBSTRING((select top 1 name
from sysobjects),1,1)) > X WAITFOR
5 --''.
This produces the following query:

SELECT email FROM user_mst WHERE
user_name='legalUser' and
ASCII(SUBSTRING((select top 1 name
from sysobjects),1,1))
> X WAITFOR 5 -- ' AND user_pwd=''
AND user_id=0
```

In this attack the `SUBSTRING` function is used to extract the first character of the first table's name. Using a binary search strategy, the attacker can then ask a series of questions about this character. In this case, the attacker is asking if the ASCII value of the character is greater-than or less-than-or-equal-to the value of X. If the value is greater, the attacker knows this by observing an additional 5 second delay in the response of the database.

Alternate Encodings

An attack intention of alternate encoding attack is evading detection. In this attack, the injected text is modified so as to avoid detection by defensive coding practices and also many automated prevention techniques. This attack type is used in conjunction with other attacks. Alternate encodings do not provide any unique way to attack an application; they are simply an enabling technique that allows attackers to evade detection and prevention techniques and exploit vulnerabilities that might not otherwise be exploitable.

To avoid this defense, attackers have employed alternate methods of encoding their attack strings (e.g., using hexadecimal, ASCII, and Unicode character encoding). Common scanning and detection techniques are not sufficient to encoded strings, thus allowing these attacks to go undetected. So at different layers of application have different ways of handling alternate encodings. The application may scan for certain types of escape characters that represent alternate encodings in its language domain. Another layer (e.g., the database) may use different escape characters or even completely different ways of encoding. For example, a database could use the expression char(120) to represent an alternately-encoded character "x", but char(120) has no special meaning in the application language's context. An effective code-based defense against alternate encodings is difficult to implement in practice because it requires developers to consider of all of the possible encodings that could affect a given query string as it passes through the different application layers. Therefore, attackers have been very successful in using alternate encodings to conceal their attack strings.

Example: In this attack, the following text is injected into the user name field: "legalUser'; exec(0x73687574646f776e) - - ". The resulting query generated by the application is:

```
SELECT email FROM user_mst WHERE
user_name='legalUser';
exec(char(0x73687574646f776e)) --
AND user_pwd='' AND user_id=
```

This example makes use of the char() function and of ASCII hexadecimal encoding. The char() function takes as a parameter an integer or hexadecimal encoding of a character and returns an instance of that character. The stream of numbers in the second part of the injection is the ASCII hexadecimal encoding of the string "SHUTDOWN." Therefore, when the query is interpreted by the database, it would result in the

execution, by the database, of the SHUTDOWN command.

XSS type attack

In the following example, we will assume that the attacker's final goal is to steal the victim's cookies by exploiting XSS vulnerability in the website. This can be done by having the victim's browser parse the following HTML code:

Example Query-

```
<script>
window.location='http://attacker/?
cookie='+document.cookie
</script>
```

This script navigates the user's browser to a different URL, triggering an HTTP request to the attacker's server. The URL includes the victim's cookies as a query parameter, which the attacker can extract from the request when it arrives to his server. Once the attacker has acquired the cookies, he can use them to impersonate the victim and launch further attacks.

II. RELATED WORKS

Existing system in practice used in development and test time to prevent or detect vulnerability attacks so that it improve programs for input validation vulnerability attacks can be reduced. Following section includes study of those techniques and also compare with our approach.

(A)Detection techniques for SQL injection Attacks-

AMNESIA(Analysis and Monitoring for Neutralizing SQL Injection Attacks)

This is the most relevant detection technique proposed by Halfond et.al.[11], author suggested that AMNESIA is the effective SQLAs detection tool. It is a model based techniques which combines both static analysis and dynamic analysis for preventing and detecting web application vulnerability at run time. Static phase is used to generate different type of query statements. Dynamic phase is used to interpret all queries before they are submitted to the database and validate each query against the statically built query models. AMENSIA technique stops all queries before they are sent to database and validates each query statement against the AMNESIA models. Queries that violate the model represent potential SQLs and thus prevented from executing on the database.

SQLGAURD [12][13]- In User input base model SQLGAURD method is useful. This method checked at runtime which is expressed as grammar that only accept legal queries.SQL Guard checks the structure of the query before and after the addition of user-input based on the model. In this approach developer should modify code to use a special intermediate library or manually insert special markers into the code where user input is added to a dynamically generated query.

SQLChecker[12] – This model uses a secret key at runtime checking so security of the approach is dependent on attackers. In SQL Check, the model is specified independently by the developer. It is similar to SQLGaurd by which Model checks as a grammar that only accept legal queries. In this case developer should have to modify code to use a special intermediate library or manually insert special markers into the code where user input is added to a dynamically generates query.

Tautology Checker [14] – This method provides an analysis framework for security. It is a static analysis and automated reasoning performs for checking any tautology statement contains in coding. The major drawback of this tool is, it having limited scope. So this tool it is not useful as much.

CANDID [14] – In java programming CANDID tool is use as dynamically for program transformation. This tool dynamically mines the programmer-intended query structure on any input and detects attacks by comparing it against the structure of the actual query issued. CANDID's natural and simple approach turns out to be very powerful for detection of SQL injection attacks.

SQL-IDS [12] - Machine learning technique includes this method. It builds a model of typical queries and matches at run time that queries that does not match with original query treat as attacks. This technique detects attacks successfully, but it depends on training seriously.

SQL Prevent [12]-This technique is consists of an HTTP request interceptor. When the original data flow is modified SQL Prevent technique is deployed into a web server. The HTTP requests are saved into the current thread of local storage. Then, SQL interceptor intercepts the SQL statements that are made by web application and pass them to the SQLIA detector module. Consequently, HTTP request from thread-local storage is fetched and checks to determine whether it contains an SQLIA. The malicious SQL statement would be prevented to be submitted to database, if it contains malicious data.

SQLRand [15] - According to the Keromytis and Boyd in SQLRand Proxy server is used between Client (Web server) and SQL server. They de-

randomized queries received from the client and sent the request to the server. Portability and security are the advantages of this de-randomization framework.

(B)Prevention techniques for SQL Injection Attacks-

WAVES [16] - WAVES a black-box technique for testing web application for SQL injection vulnerability. This technique uses a Web crawler to identify all points in a Web application that can be used to inject SQLIAs. It target on a specified list of patterns and attack techniques. WAVES then monitors the application's response to the attacks and uses machine learning techniques to improve its attack methodology.

JDBC Checker [13] –This technique is based on static analysis of web application that can reduce SQL injection vulnerabilities and detect type errors. It is uses for dynamically generated query string on basis of mismatching. As we know that most of the SQLIAs consist of syntactically and type correct queries so this technique would not catch more general forms of attacks.

SECURITYFly [12]-This is implemented for java. As compare to other tool this checks string in place of character for any suspicious information and try to sanitize query strings. This tool has a drawback that is numeric fields cannot stop by this approach. Difficulty of identifying all sources of user input is the main limitation of this approach.

SECURITY GATWAY [14] - It technique base on the filtering system that forces the input validation. By using Security Policy Descriptor Language (SPDL), developers provided specify transformation that is applied to the parameters of web application.

SQL DOM [12] - It is an object model for proposing a solution for building a secure communication environment for accessing relational databases from the OOP (Object-Oriented Programming) Languages. Due to this they mainly focus on identifying the obstacles in the interaction with the database via Call Level Interfaces.

WebSSARI [12]-Use for sanitizing input that passed through predefined set of filters. In this case static analysis to check taint flows against preconditions for sensitive functions. The drawback of this approach is that it is not necessary preconditions for sensitive function accurately expressed.

Similarly, various detection and prevention methods are being research and implemented in the past to secure web application from cross site scripting attacks. The related work includes Cross-site Scripting (XSS) Attack Detection and Cross-site Scripting (XSS) Attack prevention techniques which are mainly based on static analysis work, dynamic analysis work, static and dynamic analysis, server side solution and client side solution.

In the area of static analysis [17]Y. W Huang, F. Yu, C. Hang, C. H. Tsai, D. Lee and S. Y. Kuo describe the use of bounded model checking (BMC) for verifying Web application code. Y. Huang, S. Huang, Lin, and Tsai use number of software-testing techniques. These techniques includes black-box testing, fault injection, and behavior monitoring to web application in order to work out the presence of vulnerabilities [18]. [19]A.S. Christensen, A. Møller, and M.I. Schwartzbach describe the analysis of string expression. For this they use Java programs and checking for errors in dynamically generated SQL queries. In Taint Propagation Analysis technique they use data flow analysis to track the behavior of information flow from source to sink[20,21]. D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel and G. Vigna[22] descibe a novel approach to analysis of the sanitization process in Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications. Wassermann and Su's in their recent work [23] describe Using of untrusted scripts to detect harmful script from user given data.

In the area of dynamic analysis work, Su and Wassermann in [24] describe a successful injection attack there is a change in the syntactical structure of the exploited entity.[25]E.Kirda et al developed Noxes which is the first client-side solution to mitigate cross-site scripting attacks. Noxes acts as a web proxy, called it as personal firewall. Browser-Enforced Embedded Policies Techniques by T.Jim, N.Swamy and M.Hicks [26] developed a mechanism that modifies the browser so that it can execute only filter content to prevent injected script code from running in browsers that view the site. Interpreter-based Approaches technique was introduced to track un-trusted data at the character level and for identifying vulnerabilities that use context-sensitive string. This approach described by T. Pietraszek and C. V. Berghe[27].

In the area of static and dynamic analysis,Lattice-based Approach described by [28]D. Balzarotti, M. Cova, V. V. Felmetsger and G. Vigna,described using WebSSARI which combines static and runtime features and find security vulnerability by applying static taint propagation analysis. WebSSARI targets cross site scripting.

In the area of server side solution and client side solution [29] Rattipong Putthacharoen and Pratheep Bunyatnoparat described protecting cookies from Cross Site Script attacks using Dynamic Cookies

Rewriting technique. This is implemented in a web proxy where it will automatically rewrite the cookies that are sent back and forward between the users and the web applications. [30]Prithvi Bisht and Venkatakrishanan describes preventing mechanisms for cross site scripting attacks which is based on input validation that can effectively prevent XSS attacks on the server side. They also introduce several new XSS attacks and analyzed the reasons for the failure of filtering mechanisms in defending these attacks. They design XSS GUARD framework for preventing XSS attacks on the server side. XSS GUARD works by dynamically learning the set of scripts that a web application intends to create for any HTML request. [31]N. Ikemiya and N. Hanakawa, describe client-side mechanism for detecting malicious Java Scripts. The system consists of a browser-embedded script auditing component, and IDS that processes the audit logs and compares them to signatures of known malicious behavior or attacks. Client side cross site scripting protection using Noxes Tool [32] is a client-side Web-proxy that includes all Web traffic and serves as an application-level firewall. The approach works without attack-specific signatures.

III. PROPOSED SYSTEM

To handle SQL injection attack and Cross site scripting attack the following defenses are used for prevention. The paper identifies vulnerability attacks caused due to inputs performed by a user which are not properly validated in the web applications. We checks for any SQL Injection or XSS vulnerabilities based on an hybrid approach of query processing and regular expression checking. This approach improves the accuracy of attack detection and sanitizes the input query, so that the back end server/ database is secured. So, this approach will stop the attack before it affect the system and will provide a sanitize query to the system by classifying the input data into SQL or HTML input. The general work of the system is as follows:

1) The client sends the request to server.
2) The request is redirected to our approach query sanitization with regular expression.
3) This approach describes query processing that includes SQL injection attack and XSS attack removal services.
   (a) If http request is SQL statement then with the help of query processing it sanitizes the query with regular expression and validates the request. These validated requests then send to the web application in the server.
   (b) If http request is XSS then with the help of query processing it sanitizes the script with regular expression and validates the request. These validated requests then send to the web application in the server.
4) Else the request does not contain any malicious code, then access directly to web application server.
5) Depending on the validation results the filter on web application server decides whether to continue with the request or deny the request.

The following figure 3.1 shows the system architecture of our approach.

3.1 Block diagram of System Architecture

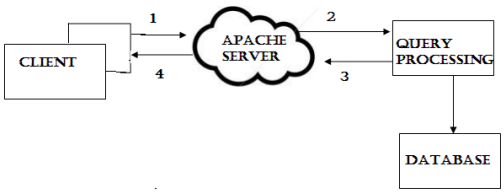

Figure 3.1. Proposed System Architecture

3.2 Flowchart of the system

The following Figure 3.2 gives the flowchart of the proposed system.

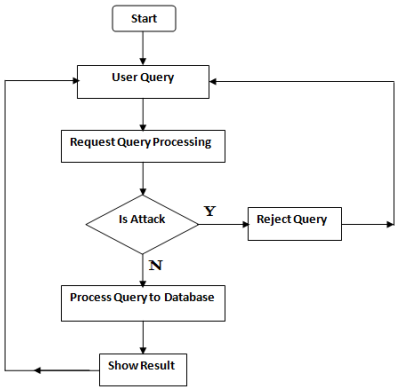

Figure 3.4 Flowchart of the system

3.3 Algorithm for hybridizing regular expression with query processing to remove SQL injections and XSS attacks

Steps:

1. Start the algorithm.
2. Accept user input in the form of any html text having scripts, tags, urls.
3. With request query processing the query will sanitize with regular expression.
4. If there is attack in request then it reject query.
   Else
   Process query to database.
5. Finally show result.
6. Repeat for each request.
7. End of the algorithm.

IV. SYSTEM IMPLEMENTATION

A web application utilizes web and browser technologies to perform tasks over a network using a web browser. The web applications are stored on the web servers, where all their data are stored.

1) This system implements the database independent SQL injection attack and XSS attack removal using regular expression algorithm using php. By using LARVEL framework we create an application.
2) The application takes the input from the URL and then by using regular expression data classifier classifies the SQL statement and script accordingly.
3) For each request the sanitizing application is executed. When the redirected request from the server reaches the sanitizing application algorithm is triggered.
4) As a first step the algorithm checks whether it is a SQL query or script which is extracted from the URL. The SQL query and script are processed using search pattern. A sequence of character that forms a search pattern is called as regular expression. This search pattern can be used for text search and texts replace operations.
5) The URL is passed onto the signature check which uses the regular expression to validate the URL.
6) Some of the following checks are done on the URL extracted from the http request.

I. Query delimeter (--)
II. White spaces
III. Comment Delimeter (/**/)
IV. Scanning for the query with
V. Dropping Meta character like (;,',<, >, %, +)

7) The validated URL are then directed back to the server.
8) Depending on the validation results the filter on web application server decides whether to continue with the request or deny the request.

V. EVALUATION

The system implements hybridizing regular expression with query processing which is detection and prevention method that remove the SQL injection and XSS attacks successfully. For the evaluation expression we used the php code for creating application. This method successfully tested on open source project. The following output screens i.e. Figure 5 shows the response of the system when a malicious input is provided in the input form. Figure 6 and Figure 7 shows outcome of malicious queries executed on application which sanitizes with our application. By using regular expression for removal of SQL Injection attack and XSS attack on independent database by firing the query which further leads to Database via URL.

Figure 5.1 E-commerce applications

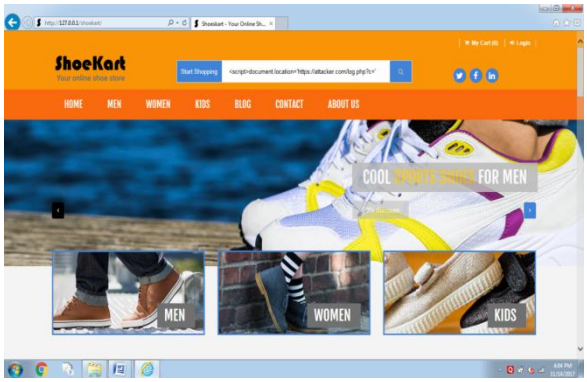

Figure 5.2 Malicious input fired to the application.

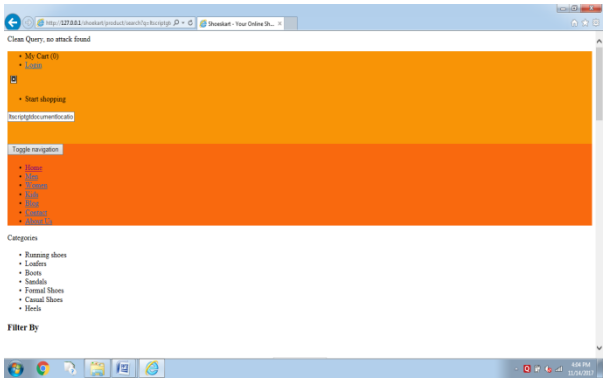

Figure 5.3 Outcome after sanitizing the malicious input fired to the application.

VI. ANALYSIS RESULT

By using the methodology (Proposed Method) used for removal of SQL Injection and XSS is successfully executed with accuracy and the analysis of methodology is shown in the below table. The hit time is also evaluated and is in ns. This section compares the detection rate of the

proposed method with other researcher's method. Table 1 and 2 shows the comparison of SQL injection detection and prevention techniques with respect to attack type. With our proposed methodology it detects and prevents all type of attack successfully and accurately. The symbol √ is used for techniques that can successfully detect all attacks of that type. The symbol × is used for techniques that is not able detect all attacks of that type. Symbol ○ shows techniques that detect the attack type only partially because of natural limitations of underlying approach.We have analyzed the system and methodology that are used to control SQL injection attacks and XSS attacks which are shown in Table3 which describes the response time for the query executed irrespective of query length with accuracy. The time taken for the response with our system was noted in nanoseconds.

Table 1. Comparison of SQL injection detection techniques with respect to attack types

Attacks / Tools	Tautology	Piggy Baked	Illeagal Incorrect	Union	Alternate encoding	Timing attack	Blind attack	Stored procedure
AMNESIA	√	√	√	√	√	√	√	x
SQLGUARD	√	√	√	√	√	√	√	x
SQL Checker	√	√	√	√	√	√	√	√
TAUTOLOGY Checker	√	x	x	x	x	x	x	x
CANDID	√	x	x	x	x	x	x	x
SQL IDS	√	√	√	√	√	√	√	√
SQL Prevent	√	√	√	√	√	√	√	√
SQL RAND	√	√	√	√	√	√	√	x
Proposed method	√	√	√	√	√	√	√	√

Table 2. Comparison of SQL injection preventiontechniques with respect to attack types

Attacks / Tools	Tautology	Piggy Baked	Illeagal Incorrect	Union	Alternate encoding	Timing attack	Blind attack	Stored procedure
WAVES	0	0	0	0	0	0	0	0
JDBC Checker	0	0	0	0	0	0	0	0
SECURITY Fly	0	0	0	0	0	0	0	0
SECURITY Gateway	0	0	0	0	0	0	0	0
SQL DOM	√	√	√	√	√	√	√	x
WebSAARI	√	√	√	√	√	√	√	√
Proposed method	√	√	√	√	√	√	√	√

Table 3. Analysis of SQL injection and XSS attack

Sr. No	Query Length	Time Taken	Accuracy
1	60	59 ns	√
2	80	51 ns	√
3	140	77 ns	√
4	144	58 ns	√
5	61	69 ns	√
6	80	51 ns	√
7	102	52 ns	√
8	140	53 ns	√
9	129	49 ns	√
10	114	54 ns	√
11	163	56 ns	√
12	191	63 ns	√
13	75	52 ns	√
14	78	51 ns	√
15	125	56 ns	√

VII. CONCLUSION

This paper presents a survey on web application attacks i.e. types of vulnerabilities and security threats within the Web application (It can be e-commerce, social networking etc.). This paper proposes improved detection and prevention of input validation attack on web applications with less time. Our proposed detection concept will help to detect and recognize XSS and SQL injection attacks and also sanitize the query for validation. It also expects that the concept will reduce the analysis time. Future work of our study will be the implementation of technique that uses method for detection and prevention of Input validation attacks like SQL injection and XSS on web application. Additionally this paper proposes improved and efficient tool that would provide web security.

REFERENCES

[1] Www.OWASP.org/index.php/XSS_Prevention_Cheat_sheet
[2] OWASP Top Ten Project - http://www.owasp.org/index.php/Top_10
[3] OWASP Code Review Guide - http://www.owasp.org/index.php/Category:OWASP_Code_Review_Project
[4] OWASP Testing Guide - http://www.owasp.org/index.php/Testing_Guide
[5] https://www.acunetix.cz/websitesecurity/cross-site-scripting/
[6] "SQL Injection/Insertion Attacks". insecure.org.
[7] SQL Injection Attacks and Defense(Book) - Justin Clarke

[8] Cross Site Scripting Wikipedia, http://en.wikipedia.org/wiki/Cross-site_scripting
[9] Cross site scripting ,accunetix,http://www.acunetix.com/website security/cross-site-scripting/
[10] Cross site scripting, Secure web development, ,http://hwang.cisdept.csupomona.edu/swanew/Code.aspx?m=XSS
[11] W. G. J. Halfond and A. Orso, "Preventing SQL injection attacks using AMNESIA," presented at the Proceedings of the 28th international conference on Software engineering,Shanghai, China, 2006.
[12] Sh. Bojken, A. Shqiponja, A. Marin, and Xh. Aleksander,"Protection of Personal Data in Information Systems",International Journal of Computer Science, Vol. 10, No. 2,July 2013, ISSN (Online): 1694-0784.
[13] Srinivas Avireddy, Varalaxhmi perumal, Narayan Gowraj, Ram Srivastava Kannan"Random4: An Application Specific Randomized Encryption Algorithm to prevent SQL Injection" 11th International conference on trust, Security and privacy in computing and communications IEEE 2012.
[14] A Survey on Detection and Prevention Techniques of SQL Injection by Harish Dehariya
[15] 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.
[16] Y. Huang, F. Yu, C. Yang, C. H. Tsai, D. T. Lee, and S. Y. Ku. "Securing Web Application Code by Static Analysis and Runtime Protection", In Proceedings of the 12th International Word Wide Web Conference, May 2004.
[17] Y. W Huang, F. Yu, C. Hang, C. H. Tsai, D. Lee and S. Y. Kuo, "Verifying Web Application using Bounded Model Checking," In Proceedings of the International Conference on Dependable Systems and Networks, (2004).
[18] Y.-W. Huang, S.-K. Huang, T.-P. Lin and C.-H. Tsai, "Web application security assessment by fault injection and Behavior Monitoring," In Proceeding of the 12th international conference on World Wide Web, ACM, New York, NY, USA, (2003), pp.148-159.
[19]]A. S. Christensen, A. Møller and M. I. Schwartzbach, "Precise analysis of string expression", In proceedings of the 10th international static analysis symposium, LNCS, Springer-Verlag, vol. 2694, pp. 1-18.
[20] V.B. Livshits and M. S. Lam, "Finding security errors in Java programs with static analysis," In proceedings of the 14th Usenix security symposium, (2005) August, pp. 271-286.
[21] N. Jovanovic, C. Kruegel and E. Kirda, "Precise alias analysis for syntactic detection of web application vulnerabilities," In ACM SIGPLAN Workshop on Programming Languages and Analysis for security, Ottowa, Canada, (2006) June.
[22] D. Balzarotti, M. Cova, V. Felmetsger, N. Jovanovic, E. Kirda, C. Kruegel and G. Vigna, "Saner: Composing Static and Dynamic Analysis to Validate Sanitization in Web Applications," In IEEE symposium on Security and Privacy, (2008).
[23] G. Wassermann and Z. Su, "Static detection of cross-site Scripting vulnerabilities," In Proceeding of the 30th International Conference on Software Engineering, (2008) May.
[24] Z. Su and G. Wassermann, "The essence of command Injection Attacks in Web Applications," In Proceeding of the 33rd Annual Symposium on Principles of Programming Languages, USA: ACM, **(2006)** January, pp. 372-382.
[25]] E. Kirda et al., "Client-Side Cross-Site Scripting Protection," Computers & Security,"Proc of 21st ACM Symposium on Applied Computing,Oct. 2009, pp. 592-604.
[26]]T. Jim, N. Swamy and M. Hicks, "BEEP: Browser-Enforced Embedded Policies," In Proceedings of the 16th International World Wide Web Conference, ACM, (2007), pp. 601-610.
[27] T. Pietraszek and C. V. Berghe, "Defending against Injection Attacks through Context-Sensitive String Evaluation", In Proceeding of the 8th International Symposium on Recent Advance in Intrusion Detection (RAID), (2005) September.
[28] D. Balzarotti, M. Cova, V. V. Felmetsger and G. Vigna, "Multi-Module Vulnerability Analysis of Web-based Applications," In proceeding of 14th ACM Conference on Computer and Communications Security,Alexandria, Virginia, USA, (2007) October.
[29] R.Putthacharoen and P.Bunyatnoparat," Protecting Cookies from Cross Site Script Attacks Using Dynamic Cookies Rewritng Technique,"Proc. of IEEE 13th International Conference on Advanced Communication Technology, Feb 2011,pp. 1090-1094.

[30] P. Bisht and V. N. Venkatakrishnan, "XSS-GUARD: Precise dynamic prevention of Cross-Site Scripting Attacks," In Proceeding of 5th Conference on Detection of Intrusions and Malware & Vulnerability Assessment, LNCS, vol. 5137, **(2008)**, pp. 23-43.
[31] N. Ikemiya and N. Hanakawa, "A New Web Browser Including A Transferable Function to Ajax Codes", In Proceedings of 21st IEEE/ACM International Conference on Automated Software Engineering (ASE '06), Tokyo, Japan, **(2006)** September, pp. 351-352.
[32] E. Kirda et al., "Client-Side Cross-Site Scripting Protection," Computers & Security,"Proc of 21st ACM Symposium on Applied Computing,Oct. 2009, pp. 592-604.
[33] T. O. Foundation. Top Ten Most Critical Web Application Vulnerabilities, 2005. `http`://www.owasp.org/documentation/top ten.html.
[34] Halfond, William G., Jeremy Viegas, and Alessandro Orso. "A classification of SQL-injection attacks and countermeasures." Proceedings of the IEEE International Symposium on Secure Software Engineering. Vol. 1. IEEE, 2006.

International Journal of Engineering Research and Applications (IJERA) is **UGC approved** Journal with Sl. No. 4525, Journal no. 47088. Indexed in Cross Ref, Index Copernicus (ICV 80.82), NASA, Ads, Researcher Id Thomson Reuters, DOAJ.

Monali Sachin Kawalkar Hybridizing Regular Expression with query processing to remove SQL Injection and XSS attacks." International Journal of Engineering Research and Applications (IJERA) , vol. 7, no. 11, 2017, pp. 13-24.