CHAPTER 4

INTELLIGENT TECHNIQUES TO PREVENT XSS ATTACKS

4.1 INTRODUCTION

Protecting database critical information and web applications from reflected and stored attacks introduced by hackers and attackers is a highly challenging issue in client-server applications. In this research work, new techniques on the browser and server side have been proposed and implemented to mitigate cross site scripting attacks. The main contribution of this thesis in preventing the two types of XSS attacks namely reflected and stored attacks is the design of new intelligent agents and production rules that effectively identify and prevent cross site scripting attacks both at browser and server.

4.2 XSS ATTACKS IN BROWSER AND SERVER SIDE

In this research work, new techniques for detecting and preventing XSS attacks have been proposed by properly monitoring the cookies and sessions using intelligent agents. For this purpose, special intelligent agents which can monitor cookies, sessions, defacing of web sites, insertion of hostile contents and phishing attacks effectively have been proposed and implemented. This is accomplished by deploying four types of intelligent agents namely cookie monitor agent, session monitor agent, content monitor agent and phishing monitor agent all of which are collectively responsible for
effective monitoring and controlling of misuses and attacks in the client side. In addition to these client side agents, two server side agents namely coordination agent and data access agent have been developed and deployed in the server side.

Therefore, in the server side, various components including the HTTP input filter component, the coordination agent, the data access agent, the filter table, the white list filter table, the data access object and the data base have been incorporated that prevent reflected and stored attacks in order to provide security in the server side.

4.3 ARCHITECTURE FOR PREVENTING XSS ATTACKS

In the proposed method, client and server side solutions are provided to mitigate cross site scripting attacks using intelligent agents. The main advantage of the server side solution is that it effectively reduces cross site scripting attacks at the server. Moreover, the client side solution prevents and takes care of cookies, sessions and phishing attacks in the client side. The proposed client cum server side solution prevents all attacks effectively and intelligently without relying on web application providers.

Since, this proposed solution is based on both client and server sides, this thesis explains the solutions provided at the client as well as server sides and then elaborates on the techniques used to filter malicious scripts in web applications on server side using rules in addition to the protection at the client side by intelligent agents. Moreover, the cross site scripting attacks are based on the possibility of inserting malicious scripts into web pages which are to be shown to other users. So a new filtering technique is necessary for filtering the malicious scripts that are injected by malicious users in web applications.
This thesis describes new intelligent agent based mechanisms to filter scripts in web applications at the server side since this has the capability to analyze all web pages which have embedded links. In this work, a rule based approach has been used to determine if a request for a resource is a local link. It is achieved by checking the referrer HTTP header and comparing the domain in the header with the domain of the requested web page. The architecture of the subsystem proposed in this thesis for detecting and preventing XSS attacks is shown in Figure 4.1.

![Architecture for Preventing XSS Attacks on Client and Server Side](image)

**Figure 4.1 Architecture for Preventing XSS Attacks on Client and Server Side**

This subsystem consists of mainly eleven major components namely, the Session Monitor Agent, the Cookie Monitor Agent, Reverse Proxy, the HTTP Input / Output tag Filter, the Coordination Agent, the Data Access Agent, the White List Filter Table, Read Filter Table, Rule Manager, Knowledge base and the Data base.

### 4.3.1 Session Monitor Agent

A session between the client and the server is created each time a connection is established. The session monitor agent uses two types of
mechanisms to monitor the application outputs. First, the session monitor agent goes directly to the screen buffer of the user session and examines the ready content. If it finds that anything is changed, it makes the changes into a package according to the transportation protocol and terminal type, and sends it to the terminal. Second, the session monitor agent builds a special communication instrument, called "pipe" and redirects the application's output to that pipe so that it can be stored and used for further analysis.

4.3.2 Cookie Monitor Agent

The cookie monitor agent monitors and maintains the cookies information in the browser so that it can prevent information leakage, incepting cookies and scripts that retrieve information from cookies. Hence, the cookie monitor agent is used to provide both security and privacy to the user queries and cookies.

4.3.3 HTTP Reverse Proxy

The reverse proxy intercepts all HTML requests and responses from client and server as well as modified web browser which is utilized to detect the script content. All the incoming requests to the web server are routed through the HTTP reverse proxy since it can deal with the request and the response appropriately. The main role of the HTTP reverse proxy is to add the necessary HTTP headers and to rewrite the existing request according to the specified rules proposed in this system. Once the request is processed, it sends back the response where the reverse proxy deletes unnecessary return HTTP headers and script code which are present in the response of the server and sends it to the requesting user.
4.3.4 HTTP Input Filter

The HTTP Input filter component is used to inspect escape comments, balance HTML tags, remove blank spaces and to handle protocol attributes from the incoming HTTP request and finally to encode this parameter. This filter encodes the data that are received as input and writes them as scripts. By using techniques such as URLEncode and HTMLEncode, it prevents the executions of malicious scripts. The output filter is used to filter the output responded by the server based on the user query. If an application itself is returning validation scripts in its documents, then the script code is removed in the final documents before it is sent to the user’s browser. Therefore, filtering is performed for checking untrusted or tainted output that is responded by the server.

4.3.5 Coordination Agent and Data Access Agent

The coordinating agent monitors the HTML input filter for the detection of inputs that are sent to the server side. It also coordinates with the rule manager and data access agent in finalizing actions. The data access agent checks the validity of user for privileges and the scripts used for accessing the database by checking whether it contains any malicious script content. If any unusual queries are found, it prevents access to the database even if they have the necessary privileges.

4.3.6 Read Filter Table

The read filter table maintains the list of present and past collections of user data’s and compares them with queries for identifying the script that contains any XSS attacks. It also compares the script with the white list filter table for identifying the white spaces in the scripts.
4.3.7 Rule Manager and Knowledge Base

The rule manager is responsible for making final decisions on the scripts. The rule manager uses the facts and rules available in the knowledge base and derives new facts using queries given in scripts. This rule manager uses forward chaining control flow for making inferences. It uses rule matching and rule firing algorithms to make final decision on the intent of the scripts.

The Knowledge base consists of basic facts about the syntax and possible semantics of queries. In addition, it has IF….THEN rules pertaining to the behavior of scripts. These rules have been generated using past knowledge, trends governing changes and by consulting with domain experts. User behaviors are also taken care of using trust computations and based on these rules optimization is performed.

4.4 IMPLEMENTATION DETAILS

The implementation of this work has resulted in significant improvement with respect to detection and prevention accuracy of the XSS attacks both on the client and server side. This has been possible because the decisions have been made using intelligent agents and rules. Hence it intelligently detects and prevents XSS attacks effectively. In this work, XSS filter provide facilities to identify the constructs that use internally a Unicode representation of strings that are vulnerable to attacks has been employed for providing effective security. Since it has the capability to transform automatically the national character set characters to the Unicode representation. The entire implementation has been carried out using the Java platform.
4.4.1 XSS Filter Component for Client and Server

In this work, filtering of malicious scripts has been carried out to prevent cross site scripting attacks on the input data which are sent to the application. This filtering is also performed on the output sent by the application to web browsers and therefore in most cases both input and output have been analyzed. To provide effective web security, the whole HTTP request which has been sent to the web application has been considered as input. In this, both the parameter values that are fed by users into HTML input fields as well as the full header have been analyzed.

When an attacker tries to use un-trusted information to mislead the application or to corrupt the scripts that are used in a web application, this system identifies them using script filters. For this purpose, the input data is normalized to specific character encoding by applying the XSS script filtering technique. Moreover, the most commonly used encoding format namely UTF-8 has been used in this work for encoding since the UTF-8 encoding technique provides a variable length encoding scheme. In addition to this, appropriate actions have been taken to avoid the problem of illegal UTF-8 character encodings by firing encoding validation rules. This is necessary since a simple XSS filter would not match dangerous characters because the lengths of the character encodings differ. The XSS script filter developed in this work performs validations on encoding and hence accepts only the characters with valid encoding.

An important observation made in this work is that, all links that are statically embedded in a web page have been safe with respect to cross site scripting attacks. This is possible due to the fact that the attacker does not directly use static links to encode sensitive user data. A cross site scripting attack succeeds if only the page has been completely retrieved by the browser and the script interpreter is invoked to execute malicious code present in that page.
4.4.2 HTML Input Filter

In this work, The HTML Input Filter has been implemented by using a HTML parser that uses pattern matching techniques. Filtering mechanism is necessary for detecting XSS attacks since the standard HTML parser libraries cannot cope with malformed HTML input. In this research work, the input parameter is first analyzed using the HTML parser to build up the HTML object tree. As usual, the harmful scripts and character encoding of input data in web applications which are indicated in the request header on the browser are removed in this implementation. In order to monitor and validate the filter, a set of coordinating agents have been employed.

The HTML input filtering process employed in this work consists of five components namely, the escape comments, balancing of HTML tags, removal of blank spaces, checking of tags and the use of white list filter table for removing white lists.

4.4.2.1 Escape Comments

This method takes the string given by the user as input and performs a pattern matching for searching of HTML comments [<! -- -- > ]. If it matches then it removes the HTML comments from the given input string. This pattern matching has been carried out using Java string handling classes and methods. The method used for pattern matching returns a string after filtering out the HTML comments. The pseudocode for illustrating the pattern matching of regular expressions for this escape comments is given in Figure 4.2.
protected String escapecomments( String s ) {
    Pattern p = Pattern.compile( "<--(--.*?--)">", Pattern.DOTALL);
    Matcher m = p.matcher( s);
    StringBuffer buf = new StringBuffer();
    if (m.find()){
        String match = m.group( 1);
        m.appendReplacement( buf, "<!--" + htmlspecialchars( match ) + "-->" );
    }
    m.appendTail( buf);
    return buf.toString();
}

Figure 4.2 Removal of Escape Comments in HTML

4.4.2.2 Balancing of HTML Tags

This technique checks whether there are any unbalanced HTML or script tags that are present in the given input string. If it finds any, then it checks whether the tags are balanced with proper OPEN and CLOSE tags. If the tags are not balanced, it removes the unbalanced tags from the given string. The Pseudo code used for balancing of the HTML tags is given in Figure 4.3.

protected String balancehtml( String s) {
    if (always_tags)   {
        s = regexReplace("<([^>]*?)(?=<|$)", "&lt;$1", s);
        s = regexReplace("(^|>)([^<\]*?)(?=>)", "$1$2&gt;&lt;", s);
        s = s.replaceAll("<>", "";
    }
}

Figure 4.3 Removal of Unbalanced Tags in HTML
4.4.2.3 Removal of Blank Spaces

This method checks whether there is any HTML or script tag that is present in the given input string. If it finds any, then it performs pattern match of it against a set of empty tags. If a match is found, then it removes the empty tags from the input string. A sample Psudocode to explain this process is shown in Figure 4.4.

```java
protected String processremoveblanks( String s ) {
    for ( String tag : removeblanks )
        s = regexReplace( "<" + tag + "(\s[^>]*>)?" + ">","", s);
    s = regexReplace( "<" + tag + "(\s[^>]*>)?/>","", s);
    return s; }
```

Figure 4.4 Removal of Blank Spaces in HTML tags

The user defined regular expression method for pattern matching is shown in Figure 4.5. The signature of this method consists of string parameters namely regex_pattern, replacement and input.

```java
protected String regexReplace( String regex_pattern, String replacement, String s )
{ Pattern p = Pattern.compile( regex_pattern);
  Matcher m = p.matcher( s);
  return m.replaceAll( replacement );}
```

Figure 4.5 Pattern Matching of Regular Expressions

The ability of a filter in correctly detecting XSS attacks strongly depends on how precisely the script detection component works in locating script content within the HTML code. The implemented method works well...
even when non-traditional ways are used for embedding script code. This work has been evaluated on the XSS Cheat Sheet having a collection of various XSS attack code snippets that cover a broad range of nuances regarding filter evasion. All the examples used for testing in this work have successfully helped to detect and prevent the XSS attacks effectively.

4.4.2.4 Check Tags

This method has been developed to check for the completeness of the given input string where it checks whether it has any script code or HTML tags that are incomplete. If it finds any such tag, then it compares it with the “White List Filter Table”, which is a predefined set of allowable and non-allowable tags and then it removes the harmful tags from the input string.

4.4.2.5 White List Filter Table

The white list filter table has been provided in this work for protecting the data from various attacks which are hard to specify based on the applications. This is necessary because most of the web applications are simple with respect to the input expected. A useful mechanism of white lists is that if new tags are available, they must be included explicitly into the list which helps to make a sound decision before the new tag is processed by the application. This white list also increases the overall security, because of its security shortcomings.

The two steps performed by this method for HTML filtering are as follows:

- Identification of the HTML elements (e.g. begin tag, paragraph text, ending tag) present in the script and
- Checking of the HTML elements which are on the list of allowed entities.
For identifying HTML elements or markup, appropriate parsing of the input is necessary. In this work, this parsing is performed by pattern matching. The HTML parsing techniques used in this work are capable of identifying the malformed HTML input. When the attackers intentionally send malformed requests to the application, the coordination agent in coordination with the rule manager and data access agent detects and prevents them in order to provide effective security.

4.5 EXPERIMENTAL RESULTS

The above implementation has been carried out for a number of sessions consisting of different durations. The results obtained on these experiments are shown in Figure 4.6.

![Analysis of session loss](image)

**Figure 4.6 Analysis of Session Loss**

From Figure 4.6, it has been observed that the percentage of session loss is reduced in all sessions in this proposed work when it is compared with the existing client side systems.

Figure 4.7 shows the results obtained by implementing client side, server side and both side monitoring system. From this, it can be seen that the
client cum server side monitoring using intelligent agents is capable of preventing more attacks in comparison with client side and server side mechanism.

![Chart showing number of attacks detected and prevented at client and server side monitoring](image)

**Figure 4.7  Number of Attacks Detected and Prevented at Client and Server Side Monitoring**

### 4.6 SUMMARY

In this chapter, a security mechanism for web applications from XSS attacks has been proposed, implemented and the results have been depicted. Extensive experimental investigations carried out in this research work show that the techniques proposed and implemented in this work are better than those of the existing systems with respect to detection accuracy and prevention.

Moreover, the experimental results obtained in this work show that the proposed system (Intelligent Agent based) enhances the security of web application by detecting and preventing most attacks successfully. The results show that the system implemented in this thesis work significantly improves the performance of capturing the scripts injected by the malicious users at browser as well as in server side.