Implementing a Prototype for Detecting Cross-Site Scripting Vulnerabilities by Using Taint Analysis

GRADUATE PROJECT REPORT

Submitted to the Faculty of
The School of Engineering & Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, TX

in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

by

Sneha Reddy Gunukula
Spring 2015

Committee Members

Dr. Mario Garcia
Committee Chairperson

Dr. David Thomas
Committee Member
ABSTRACT

Web applications are playing a vital role in our daily activities. Providing security to Web applications is the main objective of this paper. Exploiting these applications is becoming a new trend. Because of several vulnerabilities in Web applications, attackers can easily take control over the system, and can easily gain access to the sensitive information of the users. One such vulnerability is cross-site scripting. With the presence of this scripting in Web applications, malicious users can perform various operations using scripts and can install malware on the victim’s system. According to the recent study, SQL injection and XSS vulnerabilities are the most dangerous security vulnerabilities exploited in various popular Web applications, i.e. eBay, Google, Facebook, and Twitter. To solve the above problem, the proposed approach will try to develop a secure Web application and find the vulnerabilities of cross-site scripting. Then, taint analysis will be used to track the flow of tainted data in the source code of previously built JSP files. By using the taint analysis approach, the study seeks to code out the loop holes in the design source code of JSP files and generate a report stating various sink points in the code.
# TABLE OF CONTENTS

Abstract ......................................................................................................................... ii
Table of Contents .......................................................................................................... iii
List of Figures .................................................................................................................. v
List of Code .................................................................................................................. vii
List of Tables ................................................................................................................. viii
1. Background and Rationale ........................................................................................ 1
   1.1 Vulnerabilities ....................................................................................................... 1
   1.2 Cross site scripting ............................................................................................... 4
      1.2.1 Types .............................................................................................................. 5
      1.2.1.1 Non-Persistent Attacks .............................................................................. 6
      1.2.1.2 Persistent Attacks ...................................................................................... 9
      1.2.1.3 DOM Attacks .......................................................................................... 10
   1.3 XSS Protection ..................................................................................................... 11
   1.4 SQL Injection ....................................................................................................... 12
      1.4.1 Consequences ............................................................................................. 14
   1.5 Types of Vulnerabilities ....................................................................................... 15
      1.5.1 Information Leakage .................................................................................... 16
      1.5.2 Protection .................................................................................................... 15
      1.5.3 Failure to Restrict URL Access ................................................................. 15
      1.5.4 Protection .................................................................................................... 15
   1.6 Related Work ...................................................................................................... 16
2. Narrative .................................................................................................................... 19
   2.1 Problem Statement ............................................................................................. 19
   2.2 Motivation .......................................................................................................... 19
   2.3 Project Objective ............................................................................................... 19
3. Proposed Method ...................................................................................................... 20
   3.1 System Architecture .......................................................................................... 21
   3.2 Classes ............................................................................................................... 22
   3.3 Java Server Pages .............................................................................................. 23
3.4 Taint Analysis .......................................................................................................................... 24
3.4.1 Dynamic Analysis .................................................................................................................. 25
3.4.2 Dynamic Taint Analysis ........................................................................................................ 26
3.5 Parser ...................................................................................................................................... 27
3.6 Model View Controller ............................................................................................................. 27
3.7 Unified Modelling Language ................................................................................................... 28
  3.7.1 Use Case Diagram ................................................................................................................. 29
  3.7.2 Class Diagram ....................................................................................................................... 31
  3.7.3 Sequence Diagram ............................................................................................................... 33
  3.7.4 Flow Diagram ....................................................................................................................... 34
3.8 Requirements ........................................................................................................................... 35
  3.8.1 Eclipse Luna ......................................................................................................................... 35
  3.8.2 MySQL ................................................................................................................................ 36
  3.8.3 Tomcat Server ..................................................................................................................... 36
4. Implementing the Modules ......................................................................................................... 38
  4.1 Parsing .................................................................................................................................... 38
  4.2 Dynamic Taint Approach ........................................................................................................ 39
  4.3 Scanning .................................................................................................................................. 41
  4.4 User Interface .......................................................................................................................... 42
5. Testing and Evaluation .............................................................................................................. 44
  5.1 Testing Levels ......................................................................................................................... 44
    5.1.1 Unit Testing .......................................................................................................................... 44
    5.1.2 System Testing .................................................................................................................... 44
    5.1.3 Integrating Testing ............................................................................................................. 45
  5.2 Test cases ............................................................................................................................... 45
6. Expected Results and Conclusion ............................................................................................ 54
  6.1 Future Work ............................................................................................................................ 54
  6.2 Conclusion .............................................................................................................................. 54
Bibliography and References ........................................................................................................ 55
Appendix ....................................................................................................................................... 57
LIST OF FIGURES

Figure 1.1 Most Wide Spread Vulnerabilities .............................................. 2
Figure 1.2 Bar Diagram .............................................................................. 2
Figure 1.3 Non-Persistent Attack ............................................................... 6
Figure 1.4 Example of Non-Persistent XSS .............................................. 7
Figure 1.5 Executing the Script ................................................................. 8
Figure 1.6 An Example of Persistent XSS Attacks ................................... 10
Figure 1.7 DOM Based Attack ................................................................. 11
Figure 1.10 SQL Injection Attack ............................................................. 14
Figure 3.1 Block Diagram ....................................................................... 21
Figure 3.2 System Architecture ............................................................... 22
Figure 3.3 Table in Database .................................................................. 23
Figure 3.6 Architecture View of the Model View Controller .................... 28
Figure 3.7 Use Case Diagrams ................................................................. 30
Figure 3.8 Class Diagram ....................................................................... 32
Figure 3.9 Sequence Diagram .................................................................. 33
Figure 3.10 Flow Diagram ....................................................................... 35
Figure: 3.11 Architecture of Eclipse ....................................................... 36
Figure: 4.3 Working of Taint Analysis ...................................................... 40
Figure 4.6 Registration Page ................................................................... 42
Figure 4.7 File uploading ........................................................................ 43
Figure 4.8 Scanning Results .................................................................... 43
Figure 5.1 Customer Login Page ............................................................. 45
Figure 5.2 Registration Page .................................................................... 45
Figure 5.3 Registration Page .................................................................... 46
Figure 5.4 Registration Page .................................................................... 46
Figure 5.5 Registration is Not Successful ................................................ 47
Figure 5.6 Home Page ............................................................................ 47
Figure 5.7 File Uploading Page ............................................................... 48
Figure 5.8 File Uploading Page ............................................................... 48
Figure 5.9 File Uploading Page ............................................................... 49
Figure 5.10 Result Page ......................................................... 49
Figure 5.11 Test Case 1 .......................................................... 51
Figure 5.12 Test Case 2 .......................................................... 51
Figure 5.13 Test Case 3 .......................................................... 52
Figure 5.14 Test Case 4 .......................................................... 53
Figure 5.15 Test Case 5 .......................................................... 53
LIST OF CODE

Figure 1.8 Example of SQL Injection ................................................................. 12
Figure 1.9 Prepared Statements ........................................................................ 13
Figure 3.4 Example of Source and Sink Statements ......................................... 23
Figure 3.5 Example of Tainted Data ................................................................. 24
Figure 4.1 Parsing ......................................................................................... 38
Figure 4.2 Node Visitor Class ........................................................................ 38
Figure 4.4 Taint Code .................................................................................. 40
Figure 4.5 Scan ............................................................................................ 41
LIST OF TABLES

Table 5.1 Test Cases ........................................................................................................50
1. BACKGROUND AND RATIONALE

Web applications are programs which are designed to execute in the Web browser. These applications are written in programming languages like HTML, JavaScript, and CSS. Web browsers are referred as thin clients because of its potentiality to maintain and update the Web applications without the need of installing software on millions of client systems. Some popular Web applications are drop box, Webmail, eBay etc. The advantages of these applications are as follows: Due to improvement in technology, modern Web applications can conveniently exchange information between the server and the browser. Hence these applications require very little disk space. There are few disadvantages, also as follows: Many Web applications are prone to vulnerabilities because of weak programming. Some vulnerability that a raised in these applications are SQL injection, cross site scripting, Malicious File Execution, Information Leakage and Improper Error Handling etc.[10] In 2013, of all newly reported vulnerabilities XSS has been on top making it the most frequently reported vulnerability of the year. Figure 1.1 represents the graph of the most wide spread vulnerabilities in the year 2013. Among the vulnerabilities present XSS occupies the largest portion then followed by information leakage and SQL injection. When compared to other languages Perl has been observed the highest percentage i.e. 67% of XSS vulnerabilities [9]. Figure 1.2 represents the bar diagram of different languages which are affected by cross site scripting, it includes languages like Perl, Python, Ruby, and PHP.
Figure 1.1 Most Wide Spread Vulnerabilities

Figure 1.2 Bar Diagram
1.1 Vulnerabilities

Vulnerability is a loophole present in a system or in an application, which when exploited by a malicious attacker can cause a potential risk to the system, leaving the system open for various attacks. Vulnerabilities are classified into different categories:

- Hardware
- Network
- Organizational
- Software

In this project the main focus will be on software vulnerabilities. Software vulnerabilities occur because of lack of audit log or insufficient testing. Reasons for vulnerability to occur are as follows:

- Complexity: If the application is large then there are more chances for ambiguity and there will be several unintended access points in the application which leads to the exploitation of the system.
- Improper input usage: The developer of an application makes an assumption that the entire user’s input is properly validated. Because of improper input validation vulnerabilities such as XSS, SQL injection occurs.
- Error in the code: Software bugs in the program source code leaves the attacker to take control over the application and misuse the resources.
- Architecture flaws: Operating system designer implements few restrictions on the programs, so that only authorized users can access the files. If the administrator gives permission to everyone, then the hacker can introduce virus and malware on behalf of the administrator [25].
1.2 Cross Site Scripting

Cross site scripting is an application level attack, and it is also called as XSS. It targets the Web applications in which the user response is generated in the result page without proper validation of the user response. If, the input from the users is not properly validated there exist the chances of injecting malicious script into HTML pages. This attack is executed by injecting code like JavaScript, HTML, and VBScript. The attacker sets a trap for the victim to visit the exploited Web pages. When the user visits the exploited Web page, browser executes the malicious script and attacker take the advantage of the user system and performs illegal activities. The main goal of the attackers is to change the user personal settings, collect individual information like stealing their passwords, bank account details and many more. Due to these XSS vulnerability several Web applications have been exploited. For example the following are as follows.

1. On May 22 2014, eBay announced that its database has been compromised by multiple flaws which affected 145 million of users worldwide. One among the flaws is persistent cross site scripting which was responsible for hijacking users account by stealing their cookie information. The other vulnerabilities that compromised eBay are cookie reuse and account hijacking vulnerabilities [11].

2. On September 1 2014, Google app reported a flaw which forced the admin of the Google to disable the two step authentication for the existing users and modified the user settings by the hacker in such a way that all the emails are redirected to the attacker’s domain. The reason behind this flaw is a critical XSS which was injected into the JavaScript of the source code [11].
1.2.1 Types

There are three ways in which a hacker can inject malicious scripts into the Web pages. There as follows:

(1) Non-Persistent XSS attacks

(2) Persistent XSS attacks

(3) DOM based XSS attacks.

1.2.1.1 Non-Persistent XSS Attacks

It is also called as reflected cross site scripting. The main cause for this type of attack is when user input is entered in the HTML page and the same is displayed in the user’s browser without properly validating the input. Here the payload is not stored on the server, it is redirected to an HTML response page from the server without validating it first. It is the most commonly used attacks because of its flexibility. In this type of attack the hacker, uses a social engineering approach and directs the victim to click the malicious links. The hacker can send the crafted links to a victim’s system by using any one of the techniques like spam emails, malicious Web pages, social media messages. Figure 1.3 shows the steps involved in non-persistent attack.
Figure 1.3 Non- Persistent Attacks

Research: This is the first stage of the attack in this stage the attacker’s try to find the websites which are vulnerable to attack. Primarily the attacker verifies whether the user input is displayed in the HTML response page in any one of the below ways.

- Displaying the username in the HTML response page.
- Web applications have the search functionalities on the user interface and the same input is used to display on the response page.
- Usage of DOM parameter values such as a document.url [6].

Once the attacker finds out the potentially weak website they try to inject the script in the relevant areas and verify whether the script is being executed in the user web browser. Figure 1.4 and 1.5 shows an example of how an attacker can inject the script in the text field.
Figure 1.4 Example of Non – Persistent XSS.

In Figure 1.4 the user need to enter their input in the search box, if there is no proper input validation then malicious user can enter a harmful script in the given search box with his normal request as shown above.
In the above example as the web page is not properly validated user request is executed along with the script and a dialogue box is generated as shown in figure 1.5.

Social engineering: With the help of the social engineering the attacker manipulates the user to click the crafted link containing the malicious URL which injects the harmful code into the Web applications in any one of the following ways:

- Social media, which consists of messages and posts
- Spam emails with crafted links and HTML code.
Payload execution: When the victim clicks on the crafted link and the attack is successful payload gets executed in the victim’s browser and the result is addressed to the attacker’s system. The objective of the attacker is to steal cookie information and data theft [6].

1.2.1.2 Persistent XSS Attacks

It is also called as stored cross site scripting. In these types of attack the attacker injects the harmful code into the database in such a way that it is difficult to discover and hard to fix the code. When in later stage if the data is retrieved without proper validation, then a malicious code will be executed in the victim’s browser. Here the malicious code is present on the Website itself, there is no need to redirect to other links unlike the non-persistent attacks. Malicious code is stored on the Web server, victim when visits the infected page, browser will automatically run the code causing the victim system to be compromised for various attacks. This is the most dangerous attack when compared to non-persistent attacks. These types of attacks are very uncommon. Applications which have been exploited due to these types of attacks are forums or message boards, social networking sites [7]. Figure 1.6 shows an example of how persistent attack takes place in the forums.

![Figure 1.6 An Example of Persistent XSS Attacks.](image_url)
Forums or Message boards: When a vulnerable forum is identified and the attacker can insert the malicious script in the relevant areas like in the Figure 1.5 new topic and message boxes. The message posted in the forum is stored in the server when the victim clicks on the malicious content, messages are loaded on the victim’s browser and payload gets executed.

1.2.1.3 DOM-Based XSS Attacks

Document object model: It is useful in representing the objects in an HTML and other document types, every HTML document consists of DOM it represents the properties of the document in the form of objects from the point of the browser. Whenever the script runs on the client side the associated DOM values of the HTML page get executed in the browser. Figure 1.7 represents an example of DOM attack.

This type of attack occurs mainly when data is handled improperly in the HTML pages. The hacker can manipulate the objects of DOM to create an attack. The most popular scripts of this form are document.url, document.location, and document.referrer objects [8].

DOM cross site scripting is different from other types of scripting because of the following reasons:

- Generally HTML pages will be static so injecting malicious code in the HTML page is difficult not unlike the other dynamic pages.
- Since it does not have dynamic content the script is not executed in the server side browser. Therefore server side tools fail to detect the attack.
1.3 XSS Protection:

There are several techniques used to prevent XSS from occurring few techniques are discussed below.

1 Validation of input: A proper input validation mechanism should be implemented for validating all types of input. Input validation mechanism should check for syntax, rules, and types before the input is stored or displayed to the user. Invalid input should not be accepted in some cases, for example, in some situations error messages also contain invalid data. Error messages should also be validated.

2 Encoding output: A strong encoding method is used to encode all the user data depending on the output elements whether it is an XML or HTML. Rather than an encoding small
subset of user data it will be effective if the large subset of user data is encode depending upon the output.

3 Avoiding blacklist validation: For finding flaws in the user input or output blacklist validation is inappropriate because it is weak in finding the vulnerabilities and attackers can easily hack the applications. Instead of searching and replacing of few characters like “<”, “scripts” validate the entire code for vulnerabilities [5].

1.4 SQL INJECTION:

It is an insertion of an SQL query in the input data from client to the application server. If an unauthorized user is successful in injecting the SQL query he can have access to sensitive data like personal information about the users, banking details, credit card information etc. Hacker can also modify the database like creating, deleting, modifying, updating, execute the administrative authorities such as shutting down the database, giving commands to an operating system and many more. The main goal of the attacker is to affect the execution of predefined SQL commands by inserting the malicious SQL commands in the user input. This type of attacks is most frequently seen in applications like PHP and ASP because of their outdated functional interface, and less commonly seen in J2EE and ASP.NET applications because of their updated program interface.

The main reason for SQL injection is as follows:

1. If a user enters a malicious data into the database program from an untrusted source.

2. To construct dynamic SQL query.

To overcome the above problems the developer needs to follow few simple steps such as avoid writing dynamic queries and before executing the user supplied input, it is better to validate and then use the data to execute the query. Figure 1.8 represents an example of SQL injection attack.
Few simple techniques to avoid SQL injection vulnerabilities are as follows:

- Use of Prepared Statements (Parameterized Queries)
- Use of Stored Procedures
- Escaping all User Supplied Input

Unreliable sample:

```java
String query = "SELECT employee_information FROM employee_data WHERE user_name=\"+request.getParameter("UserName")\";
try {
    Statement statement = connection.createStatement();
    ResultSet results = statement.executeQuery(query);
}
```

Figure 1.8 Example of SQL Injection

The above Figure 1.8 is unsafe because the attacker can enter SQL query in the space provided for entering the username and when the query is executed without proper validation then it leads to database exploitation.

Reliable sample:

```java
String query = "SELECT employee_information FROM employee_data WHERE customer_name=?;"
PreparedStatement pstmt = connection.prepareStatement(query);
pstmt.setString(1, customername);
ResultSet results = pstmt.executeQuery();
```

Figure: 1.9 Prepared Statements

The statement used in Figure 1.9 uses the prepared statements to provide security. Prepared statements ensure that an attacker is not able to change the intent of a query, even if SQL commands are inserted by an attacker.
1.4.1 Consequences:

The consequences of SQL injection are as follows.

- **Integrity**: With the help of SQL injection attacker can change or modify the information in the database.

- **Authorization**: There is a possibility to change the authorization information present in database with the help of successful exploitation of SQL injection vulnerability.

- **Confidentiality**: As database holds a lot of sensitive data, then confidentiality is a common problem with this type of attacks.

- **Authentication**: If invalidated SQL commands are used to check the user input such as user name or password, then user profiles can be easily hacked with this type of attacks [23]. Figure 1.10 represents SQL injection.

![Figure 1.10 SQL Injection Attack](image_url)

In the Figure 1.10 an input text box is provided for the user to enter the query if the user enters malicious query and it is not properly validated then it displays all the data present in the database as shown above.
1.5 Different Types of Vulnerabilities:

1.5.1 Information Leakage: Most of the Web applications leak information such as information about the cookies, internal working due to some security issues in the Web application. Internal state of the application such as operations, inputs and the time taken to complete certain task can be known with the help of error messages. With the information obtained from error messages an attacker can launch harmful attacks on the Web applications. Some examples of detail description of error messages include

- If too much information is displayed in the error messages regarding the failure of SQL statements, database connections and so on.
- Functions which display different information when different inputs is passed. For example, when user enters an improper password, then the function should display the same error every time, but in a few cases it displays different error messages.

1.5.2 Protection:
Limit the description of error messages: Detail description of the error messages should not be displayed to the end user. Applications should return messages like 200 or 302 if an error occurs in the application. Those messages display standard error messages to the end user. The standard error message in case for incorrect credentials is either username/password is in correct instead of username is incorrect or failure to login message [25].

1.5.3 Failure to Restrict URL Access:
Some URLs are accessed only by authorized users, but hackers can gain access to such URLs and steal personal information present in those Web applications. To overcome this solution access control mechanism should be applied on Web applications so that only authorized users
can have access to the URL providing security of the information. The initial kind of attack was
called as force browsing.

1.5.4 Protection:

The following are the few techniques used to protect the system from failure to restrict URL
access.

1 Penetration test should be provided
2 Library files should be included
3 Protection from virus should be updated.
4 Security policies should be enabled.
5 Block all types of authorized file types to the normal user [25].

1.6 Related Work

There are many approaches present, for finding out the XSS vulnerabilities in the Web
applications few are discussed here.

J.Blasco (2007), proposed a multi agent scanner for detecting stored XSS vulnerabilities. In the
proposed architecture they used three different modules or agents to scan the vulnerabilities.
They are

- Web page parser agent
- A script injector agent
- The verification agent

In this approach information from one agent is used by another to scan the Web applications.
Scanning process of each stage is done concurrently with the other stages. Each module is
independent of other modules. The operations of each agent are as follows. The Web page,
parser scans the entire Web application for storing XSS injection. Second agent uses the information from the first agent to select a list of potential attacks. The verification agent verifies the list of attacks and produces a report about the scanning process [4].

Jovanovic (2006), implemented the first static tool name pixy tool to detect static XSS vulnerabilities by using a context sensitive data flow analysis. Pixy tool is a static analysis tool to find the vulnerabilities in Web applications. The drawback of this approach is the false positive rate is high and dynamic features are not detected with the help of this tool [2].

Zhang (2010) implemented a tool by using a taint analysis approach to detect vulnerabilities like XSS attacks, and SQL attacks in ASP. It uses a control flow graph to find the sink points in the data and also find tags and taint data in the source code. This also has high false positive rate [1].

Wasserman (2008) suggested a method to detect XSS vulnerability in this approach they used string analysis to track the untrusted substring value. The main disadvantage with this approach is, it cannot handle complex and dynamic code [12].

Peng Li (2010) proposed a static analysis approach to find the XSS vulnerabilities. In this approach the input from external user is marked as tainted and if this input is not properly validated, then it indicates the presence of vulnerability [13].

G.Aogosta (2012) used string analysis and symbolic execution approach, but it was not so precise in detecting the vulnerabilities from the source code of PHP [14].
2. NARRATIVE

2.1 Problem Statement:

One of the serious problems in the Web applications is the poor programming approach. Because of this approach many Web applications are exploited to vulnerabilities. As the usage of the Web applications is growing, providing security for this application’s is a challenging task. With the presence of these vulnerabilities in the Web application confidential data of the users such as banking details, personal information is at risk.

2.2 Motivation:

There are several approaches in finding out the vulnerabilities in Web applications, but most of them are applied to static pages. In most of the previous approaches in finding the vulnerabilities the main drawback was they could not handle the arbitrary complex and dynamic code. So there is a need of a procedure to find out the vulnerabilities in dynamic web pages and can handle complex data as well. If the vulnerabilities are found before the code is completely implemented, then the developer can find more ways of programming to provide security to the Web applications.

2.3 Project Objective:

The main goal of the project is to help the developer to find the loophole in the program source code before it is fully implemented.
3. SYSTEM DESIGN:

In the proposed approach a Web application is built on the model view controller architecture. It’s an Audit Website where user can upload their build JSP files which they want to scan for vulnerabilities before assembling in their complete application. In this type of application user’s need to create an account and can upload their JSP files which are needed to be scanned for vulnerabilities. Upon completion of uploading the files we apply the taint analysis approach for detecting different vulnerabilities.

Once the JSP files are uploaded the following steps take place:

- Source code is parsed into small chunks of data.
- Parsing will analyze the data flow of the application.
- Then taint analysis will be applied to the processed data.
- After processing file is scanned to generate the report.

The objective of this project is that the programmer will gather information about the various loopholes in the design source code of JSP files. Figure 3.1 represents the block diagram of the proposed approach.
3.1 System Architecture

System architecture is shown in the Figure 3.2 if the user is not a registered one then the initial step of the user is to register. If user is successful in registering then he can access the home page of the application where the user can either select to upload the files or can choose a different test case. The user personal information is stored in the database whenever the user visits the application he or she can use the same credentials to login in to the application. After the file is uploaded the results are displayed in the result page of the application.
3.2 Classes:

The following classes are implemented in the application. The functionality of each class is described as follows.

**Jspvalidator.java**

In this class, all the command line arguments are initialized like skip path, ignore tag, lib path, and report file. The ignore tag option is used to skip all the tags which are unnecessary to validate. Skip path is used to skip the files and directories which are separated by comma. Lib path option is used to know the absolute path of the folder containing additional dependencies. Report file argument is used to write the output to the specified location. Validator class parses the given JSP files and calls the node visitor class.
Node visitor.java

Node visitor class visits each and every node and finds the unsafe statements in the JSP file. It skips the tags which are safe and writes the unsafe expression to the output file.

Databaseconnection.java

This class helps to create the database for storing the user details and whenever the user enters incorrect details an exception is thrown stating in correct username or password. Tables are created to store the information about the users like username, email, phone number, location.

Figure 3.3 shows the registration table which is used to store the user details.

<table>
<thead>
<tr>
<th>user_id</th>
<th>user_name</th>
<th>password</th>
<th>mobile</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sneha</td>
<td>sneha</td>
<td>9393939</td>
<td><a href="mailto:abc@gmail.com">abc@gmail.com</a></td>
</tr>
<tr>
<td></td>
<td>priya</td>
<td>priya</td>
<td>5467895</td>
<td><a href="mailto:asb@gmail.com">asb@gmail.com</a></td>
</tr>
<tr>
<td></td>
<td>123</td>
<td>sfdsg</td>
<td>4562</td>
<td><a href="mailto:dfgdb@gmail.com">dfgdb@gmail.com</a></td>
</tr>
<tr>
<td></td>
<td>abc</td>
<td>shgf1786</td>
<td>78412</td>
<td><a href="mailto:dfgfd@gmail.com">dfgfd@gmail.com</a></td>
</tr>
<tr>
<td></td>
<td>krishna</td>
<td>krishna</td>
<td>4562</td>
<td><a href="mailto:dfgfd@gmail.com">dfgfd@gmail.com</a></td>
</tr>
<tr>
<td></td>
<td>(NULL)</td>
<td>(NULL)</td>
<td>(NULL)</td>
<td>(NULL)</td>
</tr>
</tbody>
</table>

Figure 3.3 Table in the Database

3.3 Java Server Pages

JSP’s are used to create dynamic web pages. Developers with the help of JSP tags insert java code in an HTML pages. Input from the user can be collected with the help of these pages. JSP tags are used to pass the control between the pages and also helpful in retrieving the information from the database. There are a variety of technologies that exist apart from JSP they are as follows:

- Active server pages
- Pure servlets
• Static HTML

• Server–side includes.

Following are the few JSP pages implemented in the application.

**Actionlogin.jsp:**

Login JSP takes username and password as parameters and validates against the database if the values are true then it give access to the user to enter into main page.

**Actionregister.jsp:**

If all the necessary fields are filled then it returns a true value which indicates of successful registration.

**Advantages of JSP**

• Better performance

• JSP can combine with servlet to handle business logic.

• Portable in nature.

### 3.4 Taint Analysis

Taint checking is one of the components present in some computer languages like Perl, ruby etc. This mechanism provides security by preventing attackers from executing the malicious code on the target systems. Taint checking provides security for most Web applications which were affected by vulnerability like XSS. The working principle behind this approach is when a variable is modified by an external user then this variable possesses a security threat to the application. If, this variable sets another variable in the program, then the second variable is also considered as a potential risk to the application. If these variables are used to execute any harmful scripts in the code, then the taint tool reports the tainted variable in the program code. It
is a type of information flow analysis, which checks each variable in the code to find out the tainted data. As it is helpful in detecting vulnerability in the Web applications, it is very popular among industries and research communities [15].

3.4.1 Dynamic Analysis:
Dynamic analysis is becoming more popular and fundamental tool in the field of computer security. It is simple and helps the users to develop precise security at run time. There are two techniques of dynamic analysis, which are commonly used, they are as follows:

- Dynamic Taint analysis
- Dynamic forward symbolic execution.

Dynamic Taint analysis: In this technique the program is executed to observe the parameters which are affected by already defined taint sources for example user input.

Dynamic Forward Symbolic Execution: Logical formula for describing a program execution path is automatically built on this type of technique. These two techniques are used in a wide variety of applications, some of them are as follows:

i) Vulnerability detection: Most web applications are prone to vulnerability attacks. This can be solved by using dynamic taint analysis. During the execution of the program dynamic taint analysis looks for inappropriate input. With the help of this technique we can prevent SQL, XSS, and code injection vulnerabilities.

ii) Automatic Input Filter Generation: To generate automatic input filters we use a forward symbolic execution technique. Filters generated by using a forward symbolic technique have a high accuracy rate.
iii) Malware Analysis: To detect malware both taint and forward techniques are needed. It can be achieved by analyzing the data flow in the malware binary [22].

3.4.2 Dynamic Taint Analysis

The objective of the dynamic taint analysis is to track the information between the source and sink points in the program. Source statement refers to the users input, for example, HTTP request parameter. Whereas sink is defined as a statement which uses the user input and generates the response for example, it includes code which execute scripts, HTML response page, SQL query. Sample code representing source and sink statements are as follows.

1. <?php
2. $id=$_GET['customer_id'];
3. $username=$_GET['customername'];
4. $color=$_GET['color_type'];
5. <HTML>
6. <script>$id='<?php echo $id;?>';</script>
7. <h1 style="color:<?php echo $color;?>">Welcome : <?php echo $customername;?></h1>
8. </body>
9. </HTML>

Figure 3.4 Example of Source and Sink Statements.

In the sample code shown in figure 3.4 the statement’s 2, 3, 4 represent the source statements and statements 6,7 represent the sink statements.

1. s := 3*get input (·)
2. y := 6 + s
3. goto y

Figure 3.5 Example of Tainted Data

In figure 3.5 get input is given by the user so it is marked as tainted .As it is being multiplied by three and stored in variable ‘s’ then the line one is tainted. The tainted variable, ‘s’ is used in the second line, then variable y is also considered as tainted. Similarly third line is also tainted as it uses the tainted variable y [22].
3.5 Parser

It is a feature of software which usually takes text as input and builds a syntax tree like structure of the input, and checks for grammar of the processed data. Depending on the input to the parser the usage of it varies in data languages, it can be used for reading the program files like HTML, XML text. In some cases it acts as a component of a compiler for interpreting the programming languages into some binary form. [17].

3.6 Model View Controller

This is a type of architecture used to implement user interface for Web applications. It consists of three modules:

i) Controller: User input is passed into these modules and it converts the text or input in the command form and passes to the next module that can be model or view.

ii) Model: It is responsible for managing the logic, data and the rules in the application. It also updates the output to the view module.

iii) View: It is used to display the output in the form of charts, diagrams, tables depending on the requirement of the users [16]. Figure 3.6 represents the functionality of the model view controller architecture.
3.7 **Unified Modelling Language:**

To describe a system design in the software field, a general purpose language called Unified Modelling Language (UML) is used. The main goal of the UML diagrams is to provide the users with a standard vision of the system and it also helps to understand the interaction between the different individual software components in the system. There are two different types of UML diagrams as follows

1. Static
2. Dynamic

Static Diagrams: It is also called as structural diagram and it mainly focuses on relationships, operations, objects, and attributes of the system. It includes class, object, component, and deployment diagrams.
Dynamic Diagrams: It is also called as a behavioral view of the system and it mainly focus on the internal state of objects and its association. It includes use case, state, interaction, sequence diagrams [19].

UML diagrams can be used in the following fields: science and research, telecommunications, defense etc.

3.7.1 Use Case Diagram:

Among the behavioral diagrams, use case is the most basic one. A use case diagram represents a graphical view of the actors and functions of the system. It also helps users to understand the interaction between the actors and functions which takes place in the process or system [20]. Figure 3.7 represents the use case diagram
The user here might be anyone like the developer, analyst, system manager who wants to interact with the system. The user firsts need to register once user is successful in registering then user can login to the main page where user needs to perform an action like uploading the files to check for vulnerabilities. Once the file is uploaded user can check the result. Additionally user can check for different test cases.
3.7.2 Class Diagram

Class diagrams are the most frequently used diagrams in object oriented languages. It gives the user an overview of the interaction taking place between the different classes in the system. Basically, this diagram has three sections the first section represents the name of the class, the second section represents the attributes and the last section consists of the functions necessary to implement the logic. The relationship between the classes can be displayed with the help of arrows [20]. Figure 3.8 represents the class diagram. The classes used in the approach are jspvalidator class, nodevisitor class, jspoptions class. The main functions of validator class includes doclean up operation, do validate pages and scan files and it is directly associated with node visitor class and jsp validator class. Node visitor class consists functionalities like visit jsp body, visit custom tags, visit unsafe expression, visit check attribute, visit un interpreted class. And is having a direct association with text file class. jspcoption class is having a realization relationship with option class the functionalities of jspc option class are implemented in the option class. jsp validator is having multiple functionalities or responsibilities associated with the text file class text file class consists of operations to write the results into the output file.
Figure 3.8 Class Diagram
3.7.3 Sequence Diagrams

It is the most popular among the dynamic modelling and represents the flow of logic of the system. Sequence diagrams are used to design and analysis the process. These diagrams are typically used to the model logic of services, methods and usage scenarios. It is also called as event scenarios. Parallel vertical lines are used between the process this represents the lifeline of the process in the system, and arrows are used to exchange the messages between different processes [18]. Figure 3.9 represents the sequence diagram first the user uploads the jsp files and then parsing is applied on the file to create nodes of the JSP document after parsing the file is validated with the help of dovalidate class and pass the control to node visitor class to visit each and every node and generate a result page describing the vulnerabilities present in the file.

![Sequence Diagram](image)

Figure 3.9 Sequence Diagram
3.7.4 Flow Diagram:

Flow diagram represents the flow of a system. The initial step in the application is to start the server and once the server starts successfully then user has two options to select either to upload the file or select the different cases. Depending upon the choice of the user, the flow continues as shown in the figure 3.10.

start

Start tomcat server

Select different vulnerability cases

false

Upload JSP files

Parsing

Compiling

Scanning

result

XSS Vulnerability

SQL Vulnerability

LFI Vulnerability

RFI Vulnerability

stop
3.8 Requirements

The following are the software requirement’s which are necessary to build the application. It includes the platform on which the application is built the database and the server. More detail description of the requirements is as follows.

3.8.1 Eclipse Luna: Platform of the eclipse is built around the concepts of plugins. Plugins are the structured data which is used to increase the functionality of the system. Code libraries, platform extensions will be helpful in increasing the functionality. There are several subsystems present in the eclipse platform and each acts as a plugins and implements some main functions. Figure 3.11 represents the architecture of the eclipse platform.

Eclipse SDK: It consists of basic platform and additionally two tools which will be helpful in plugin development.

Java Development Tool: Java development tool (JDT) is useful in implementing the java environment.

Plug-in Developer Environment (PDE): It adds specialized tools that streamline the development of plug-ins and extensions.

Workbench: This is mainly assigned for desktop applications. The goal of this subsystem is to integrate and control the resources.

JFace : JFace provides the tool kit to develop user interfaces. It consists of several classes to implement the user interface.

Standard Widget Toolkit: Standard Widget Toolkit (SWT) consists of portable API and strict underlying native OS GUI platform [21].
3.8.2 MYSQL

Relational database is the most widely used databases because it is a free software. In web applications the popular choice will be MYSQL database.

3.8.3 Tomcat Server:

It was developed by apache and it acts as a container to the servlets, it is a standalone server for Web applications like servlets, HTML and JSP or it can be plugged in to a web server like apache , Microsoft server etc. Web applications are the Web sites which usually consist of several modules such as JSP, servlets, Web pages. The objective of the tomcat is to organize all these multiple modules into a single directory for every Web application. For creating the servlet tomcat need two different directories the first one is called as the deployment directory in which
live code is kept and the second one is the development directory this directory helps the developer to write and partially debug the code. To avoid the conflicts between the Web applications we use packages. Whenever a user submits an HTML form, then servlet is found based on the URL and generates the response either in HTML page or servlet forwards the response to the JSP page and this page gets embedded into HTML page and finally tomcat server displays the HTML page to the user [24].
4. IMPLEMENTION OF THE MODULES:

4.1 Parsing:

In servlet container there is no such mechanism which implements the JSP. So the job of the servlet container is to convert the JSP page into the servlet. Automatically servlets gets registered to handle different URLs. This transformation is done by jasper compiler on tomcat server with the help of this library function (org.apache.jasper.complier). With the help of the parser class (org.apache.jasper.compiler.Parser) JSP pages are parsed. In the end generator class generates the servlet code by using (org.apache.jasper.Generator) classes. Later the parsing of JSP file is done through parser class. The Parser class parses the page and outputs a list containing nodes. Node consists of internal representation of data of a JSP document. Node visitor traverses the node in the recursive order which is implemented by the generator class to produce servlet code.

```java
parser = new BasicParser();
cl = parser.parse(opts, args);
```

Figure 4.1 Parsing

In the above Figure 4.1 an instance for parser is created and it take options and arguments as parameters here the options include skip path, lib path, ignore tags and report file. With the help of this parser object JSP pages are parsed.

```java
public void visit(JspBody n) throws JasperException
{
   System.out.println("visit1");
   if (n.getBody() != null)
      n.getBody().visit(this);
}
```

Figure 4.2 Node Visitor Class
The node visitor class shown in figure 4.2 creates an instance to visit each and every node of the JSP file and executes each node it visits. Different classes are created to handle different type of exceptions.

4.2 Dynamic Taint Approach:

When propagating through the code the input which is derived from the source such as request parameters, HTML request should be marked as tainted. Strings which are derived from the tainted source are also marked as tainted. To track the flow of the tainted data in the code attach a flag with every string. The flag is set when a string is referred from the source method this tainted flag is propagated through a tainted string by using operations like string concatenation, case conversion etc. Implementation of taint analysis requires specification of source and sinks, strings which are referred from source elements and marked as tainted. Exception is to be raised when a tainted string is used in the sink method [22]. Java.lang.string class is used to find the tainted information from the strings. Java string classes are loaded with predefined module called as class loader. With the help of this java string classes add a Boolean tag to a class to indicate whether its value is tainted or not. Even if one parameter of the string is tainted with the java string class considers the whole string value as tainted. If the string is source it will mark the return value of the string as tainted and if the value is sink, then the argument to the sink method is marked as tainted. Figure 4.3 represents the internal functionality of the taint analysis.
Figure 4.3 Working of Taint Analysis.

```java
private boolean skipTag(String qName) {
    System.out.println("skip tag");
    boolean retVal = false;

    if (null == qName) return true;
    if (null == _arrSkipTags) return false;

    for (String tagName : _arrSkipTags) {
        if (tagName.equalsIgnoreCase(qName)) {
            retVal = true;
            break;
        }
    }
    return retVal;
}
```

Figure 4.4 Taint Code

The Figure 4.4 represents the taint code which returns true if the string is tag name and if the string is skip tags then it returns a false value.
4.3 Scanning:

Scanner is a program which helps in finding out the security vulnerabilities and design errors in Web applications. It follows the black box methodology. Black box is considered as an object or system, it only considers inputs and outputs without taking into consideration about its internal working. Jar scanner elements under the tomcat server consists of components which are used to scan the web applications for JAR files. \texttt{org.apache.tomcat.util.scan.StandardJarScanner} is used to implement the jar scanner.

\begin{verbatim}
    lstPages = scanFiles(new File(pageRoot));
\end{verbatim}

Figure 4.5 Scan Code

Figure 4.5 represents the scan code in which \texttt{scanFiles} take \texttt{pageRoot} as an argument and scans the file located in that path.

Limitations:

1. Most of the free tools are not updated to the latest vulnerabilities in the industry.

2. Logical flaws cannot be detected with the help of this tool.

3. Even the technical flaws are difficult to find if the correct direction is not provided by the application.

Strengths:

1. It will be helpful in detecting several vulnerabilities in web applications like XSS, SQL injection etc.

2. Not language dependent.
4.4 User Interface:

The application is a Web based application so here the first step is to create a register page for the users who wants to validate their JSP files for vulnerabilities. After successful registration the users are navigated to the main page where they need to enter their username and password. Once the user is logged in he can click on various links which contains the description of several vulnerabilities or he can upload the JSP files to check the bugs in the application. Once the file is successfully uploaded then a link is provided to check the scanning results. When the user clicks on the link the result is generated in a tabular form representing the causes of vulnerabilities.

1. The first step user needs to perform is the registration. The following screen shot 4.6 shows the registration user interface.

![Registration Page](image)

Figure 4.6 Registration Page.

2. Later the users can browse the file and can upload for scanning. Screen shot 4.7 displays the interface where user need to browse the files which he need to scan.
3. After scanning the JSP files result is generated where user can see the report of scan to find the loopholes in the application. Following screenshot 4.8 displays the interface of the generated results.

Figure 4.7 File Uploading

Figure 4.8 Scanning Results.
Testing is a key to success. In software development cycle testing plays a major role testing helps to find the errors in the products which are not known in the initial stage of development. In testing phase a set of inputs is passed to the system and the output is evaluated to check whether the system is meeting its requirement’s as expected. Testing helps the developer to find the loopholes in the application before it is fully completed. The goal of testing is to find errors in the applications or the programs and report them to the user, such that user can improve the quality of the product by readdressing all the issues.

There are two types of testing methods available in general static and dynamic testing:

Static testing: In this type of testing the user will mainly focus on the syntax, program structure, and conduct walk through and inspections. Static testing is useful for verification.

Dynamic Testing: In this type of testing program or applications are running in a particular environment before the entire application is developed such that to check or focus on particular modules of the code. Dynamic testing is useful for validation.

5.1 Testing Levels:

5.1.1 Unit Testing: It is also called as component testing it is used to verify the functionality of a specific section of the code and it is written by developers one function can have multiple test cases it is generally implemented in the construction phase.

5.1.2 System Testing: It is also called as end to end testing it is tested on complete application not unlike the unit testing for example, testing on the interface like validating the username, password and validating the other input fields etc.
5.1.3 Integrating Testing: Testing the individual modules will be not efficient in finding all the errors in an application if testing should be effective then we need to integrate all the components into a single module and testing should be performed on it.

5.2 Test Cases:

1. First the user should register to upload JSP files. Once the user creates valid username and password, user is navigated to the main page. Figure 5.1 and 5.2 represents the login and register pages.

![Customer Login Page](image1)

**Figure 5.1 Customer Login Page**

![Registration Page](image2)

**Figure 5.2 Registration Page**
2. If the user does not enter any of the fields such as email id, username, mobile, password following dialogue box appears and a message is displayed to the user as shown in screenshot 5.3

![Registration Page](image)

**Figure 5.3 Registration Page**

3. Once all the required fields are correctly filled, then registration is said to be complete and following screenshot 5.4 gives the visionary of the user interface after the successful registration process.

![Registration Process](image)
Figure 5.4 Registration Page

4. Suppose if user provided any invalid input like for example too many digits in the phone number, then the registration is said to be incomplete or user need to register again. Screenshot 5.5 shows the user interface of unsuccessful registration.

Figure 5.5 Registration is Not Successful

5. After user creates a valid username and password he can access the main page where he should use his credentials to login to upload his files. A proper username and password is required to login to the page and if the user forgot to enter the password, then following message is displayed to the user. Screen shot 5.6 represents the scenario where a user forgets to enter his password.
6. After the successful login user is navigated to the main page where he can browse and upload his files to check for vulnerabilities as shown in screenshot 5.7.

7. If no file is selected to scan, then following message is displayed to the user as shown in the screen shot 5.8.

8. After the file is uploaded successfully a link is provided to check for the vulnerabilities present in the file. Screen shot 5.9 shows the link to check the scanned result.
9. If file is uploaded successfully, then scan result is generated to check the result user need to click on the link scan result to find the errors in his JSP pages. Report is generated as follows.

In the following screen shot 5.10 report of the scanning is displayed the report is generated in a tabular form. It consists of the file name, type of vulnerability, the line at which the vulnerability is occurring and the it also displays the statements at which error has occurred. Table 5.1 represents more test cases with tainted data.
<table>
<thead>
<tr>
<th>S.NO</th>
<th>Input</th>
<th>Test Case</th>
<th>Expected Result</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>JSP File</td>
<td><code>&lt;c:choose&gt;</code>&lt;br&gt;&lt;br&gt;  &lt;c:when test=&quot;${casesForm.institutionRepresents == 'C'}&quot;&gt;  &lt;c:set var=&quot;cdt&quot; value=&quot;${checked}&quot;/&gt;&lt;/c:when&gt;&lt;br&gt;  &lt;c:when test=&quot;${casesForm.institutionRepresents == 'D'}&quot;&gt;  &lt;c:set var=&quot;dbt&quot; value=&quot;${checked}&quot;/&gt;&lt;/c:when&gt;&lt;br&gt;  &lt;/c:choose&gt;`</td>
<td><code>casesForm.institutionRepresents == 'C'&lt;br&gt; checked&lt;br&gt;&lt;br&gt;  </code>casesForm.institutionRepresents == 'D'&lt;br&gt; checked`</td>
<td>successful</td>
</tr>
<tr>
<td>2</td>
<td>JSP File</td>
<td><code>&lt;input type=&quot;hidden&quot; value=&quot;&lt;%=productIDs %&gt;&quot;&gt;</code></td>
<td><code>productIDs</code></td>
<td>successful</td>
</tr>
<tr>
<td>3</td>
<td>JSP File</td>
<td><code>input type=&quot;hidden&quot; name=&quot;newAnticsrfToken&quot; id=&quot;newAnticsrfToken&quot; value=&quot;&lt;%=newAnticsrfToken %&gt;&gt;\n&lt;br&gt;  </code>&lt;input type=submit value=&quot;submit&quot;&gt;`</td>
<td><code>newAnticsrfToken</code></td>
<td>successful</td>
</tr>
<tr>
<td>4</td>
<td>JSP File</td>
<td><code>&lt;div&gt;&lt;br&gt;  &lt;label for=&quot;txtBody&quot;&gt;Message :\&lt;/label&gt;&lt;br&gt;  &lt;textarea rows=&quot;10&quot; cols=&quot;72&quot; id=&quot;txtBody&quot; name=&quot;txtBody&quot;&gt;${request.name}&lt;/textarea&gt;&lt;br&gt;  &lt;/div&gt;&lt;br&gt;  </code>&lt;input type=&quot;hidden&quot; name=&quot;newAnticsrfToken&quot; id=&quot;newAnticsrfToken&quot; value=&quot;&lt;%=newAnticsrfToken %&gt;&gt;\n&lt;br&gt;`</td>
<td><code>request.name&lt;br&gt;&lt;br&gt;  </code>newAnticsrfToken`</td>
<td>successful</td>
</tr>
</tbody>
</table>
The Figure 5.11 represents the screenshot of the sample test case 1 the file name is test 1 and in the result table it is displaying the vulnerabilities and the line number at which it is occurring.

Figure 5.11 Test Case 1

The Figure 5.12 represents the screenshot of the sample test case 2 the file name is test 3 and in the result table it is displaying the vulnerabilities and the line number at which it is occurring. Here the JSP type is Scriplet.

Figure 5.12 Test Case 2
The Figure 5.13 represents the screenshot of the sample test case 3 the file name is case 4 and in the result table it is displaying the vulnerabilities and the line number at which it is occurring. Here the JSP type is Scriplet.

The Figure 5.14 represents the screenshot of the sample test case 4 the file name is mixed and in the result table it is displaying the vulnerabilities and the line number at which it is occurring. Here the JSP type is Scriplet and Enline html.
The Figure 5.15 represents the screenshot of the sample test case 5 the file name is test 2 there are no vulnerabilities detected in the file.
6. RESULTS AND CONCLUSIONS

In this paper Web applications are scanned for vulnerabilities. After finding the vulnerabilities, application will generate a report, stating various sink points in the code. So at the end, programmer can figure out all the vulnerabilities that are to be fixed in their JSP files.

6.1 Future Enhancements

To enhance the project, a framework can be developed such that it automatically eliminates the different kinds of vulnerabilities from the source code of Web application. In this application only one type of vulnerability has been detected in future SQL injection can also be implemented. While developing a prototype for SQL injection log file was not generated so this can be enhanced in future.

6.2 Conclusion

There are many approaches for detecting the XSS, SQL vulnerabilities in Web applications. Unfortunately, still many Web applications are being exposed to these types of vulnerabilities. In the proposed approach, by using a taint analysis technique a Web application is created. With the help of this application source code of built Web applications will be scanned and find the sink points in the code which are the major causes for different type of vulnerabilities.
BIBLIOGRAPHY AND REFERENCES


8. APPENDIX: CODE SNIPPETS

```java
public void visit(CustomTag n) throws JasperException
{
    System.out.println("visit4");
    visitAttributes(n);
    if (n.getBody() != null)
        n.getBody().visit(this);
}
```

Figure 8.1 Node Visitor Class for CustomTag

```java
private void visitAttributes(Node n)
{
    System.out.println("visit attributes");
    String qName = null;
    String value = null;
    Attributes attrs = null;

    qName = n.getQName();
    if ("spring:message".equalsIgnoreCase(qName))
    {
        checkTextAttribute(qName, n);
        return;
    }
}
```

Figure 8.2 Node Visitor Class for VisitAttributes.