Detection of SQL Injection and XSS Vulnerability in Web Application

Priti Singh, Kirthika Thevar, Pooja Shetty, Bushra Shaikh

***Abstract*— The increasing dependence on web applications has made them a natural target for attackers. Among these attacks SQL Injection Attacks (SQLIA) and Cross-Site Scripting attacks are the most prevalent. Our SQL Injection detection method is based on the design of a detection tool for the HTTP request send by clients or users and look for attack signatures. The proposed filter is generic in the sense that it can be used with any web application. Finally we test our proposed security mechanism using the vulnerability scanner developed by us as well as other well-known scanners. Our approach for Cross-Site Scripting detection method describes the possibilities to filter JavaScript in Web applications in server side protection. Server side solution effectively protects against information leakage from the user's environment. Cross-Site scripting attacks are easy to execute, but difficult to detect and prevent.[1]**

***Index Terms*— SQL Injection, Cross-Site scripting, scanner, signature, vulnerability.**

I. INTRODUCTION

The security of Web applications has become increasingly important in the last decade. More and more Web based enterprise applications deal with sensitive financial and medical data, which, if compromised, in addition to downtime can mean millions of dollars in damages. It is crucial to protect these applications from hacker attacks. A great deal of attention has been given to network level attacks such as port scanning, even though, about 75% of all attacks against Web servers target Web-based applications, according to a recent survey. Traditional defense strategies such as firewalls do not protect against Web application attacks, as these attacks rely solely on HTTP traffic, which is usually allowed to pass through firewalls unhindered. Thus, attackers typically have a direct line to Web applications. Many projects in the past focused on guarding against problems caused by the unsafe nature of C, such as buffer overruns and format string vulnerabilities. However, in recent years, Java has emerged as the language of choice for building large complex Web-based systems, in part because of language safety features that disallow direct memory access and eliminate problems such as buffer overruns. According to the OWASP survey made SQL Injection and XSS are the top two vulnerabilities found in web application[1,2].

Priti Singh, IT Department, Mumbai University SIES Graduate School Of Technology, Nerul, India
Kirthika Thevar, IT Department, Mumbai University SIES Graduate School Of Technology, Nerul, India
Pooja Shetty, IT Department, Mumbai University SIES Graduate School Of Technology, Nerul, India
Bushra Shaikh, IT Department, Mumbai University SIES Graduate School Of Technology, Nerul, India

II. D-WAV: DETECTION OF WEB APPLICATION VULNERABILITY

A. What is SQL Injection Attack?

A SQL injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to effect the execution of predefined SQL commands.[1]

SQL Injection Types:

1) UNION ATTACK:

By this technique, attackers join injected query to the safe query by the word UNION and then can get data about other tables from the application. Suppose for our examples that the query executed from the server is the following: SELECT Name, Phone FROM Users WHERE Id=$id By injecting the following Id value: $id= 1 UNION ALL SELECT creditCardNumber, 1 FROM CreditCarTableWe will have the following query: SELECT Name, Phone FROM Users WHERE Id= 1 UNION ALL SELECT creditCardNumber, 1 FROM CreditCarTable which will join the result of the original query with all the credit card users[1].

2) BLIND INJECTION:

Sometimes developers hide the error details which help attackers to compromise the database. In this situation attacker face to a generic page provided by Developer, instead of an error message. So the SQLIA would be more difficult but not impossible. An attacker can still steal data by asking a series of True/False questions through SQL statements. Consider two possible injections into the login field: SELECT accounts FROM users WHERE login='doe' and1 =0 -- AND pass = AND pin=O SELECT accounts FROM users WHERE login='doe' and1 = 1 -- AND pass = AND pin=O

If the application is secured, both queries would be unsuccessful, because of input validation. But if there is no input validation, the attacker can try the chance. First the Attacker submit the first query and receives an error message because of "1=0". So the attacker does not understand the error is for input validation or for logical error in query. Then attacker submits the second query which always true. If there is no login error message, then the attacker finds the login field vulnerable to injection.[1]

3) ERROR-BASED ATTACK:

An Error based exploitation technique is useful when the tester for some reason can't exploit the SQL injection vulnerability using other technique such as UNION. The Error based technique consists in forcing the database to perform some operation in which the result will be an error. The point here is to try to extract some data from the database and show it in the error message. This exploitation technique can be different from DBMS to DBMS. Consider the following SQL query: SELECT * FROM products WHERE id_product=$id_product Consider also the request to a script who executes the query above: http://www.example.com/product.php?id=10 The malicious request would be (e.g. Oracle 10g): http://www.example.com/product.php?id=10||UTL_INADDR.GET_HOST_NAME((SELECT user FROM DUAL))--

In this example, the tester is concatenating the value 10 with the result of the function UTL_INADDR.GET_HOST_NAME. This Oracle function will try to return the host name of the parameter passed to it, which is other query, the name of the user. When the database looks for a host name with the user database name, it will fail and return an error message like: ORA-292257: host SCOTT unknown[3]

B. What is Cross-Site Scripting?

Cross-Site Scripting (XSS) attacks occur when:

1. Data input in Web application through an untrusted source, mainly a web request.
2. The data is included in dynamic content that is sent to a web user as HTTP Response without being validated for malicious script.

The malicious content sent to the web browser is a piece of JavaScript, but it may also include HTML or any other type of code that the browser may able to execute. The variety of attacks based on XSS is very vast, but commonly they include transmitting confidential data like cookies or other essential session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the appearance of the vulnerable site.[4]

C. Existing System

Existing system used to identify bugs in source code often return large numbers of false positive warnings to the user. True positive warnings are often buried among a large number of distracting false positives. By making the true positives hard to find, a high false positive rate can frustrate users and discourage them from using an otherwise helpful tool. The use of historical data mined from the source code revision history can be useful in refining the output of a bug detector by relating code flagged by the tool to code changed in the past.

D. Proposed System

In our proposed system, we propose a SQL Injection Detection method which is based on designing a Detection tool for the HTTP request send by clients or users and look for attack signatures. The proposed tool is generic in the sense that it can be used with any web application. Finally we test our proposed security mechanism using the vulnerability scanner developed by us as well as other well-known scanners. Our second approach for Cross-Site Scripting detection method describes the possibilities to filter JavaScript in Web applications in server side protection. Server side solution effectively protects against information leakage from the user's environment. Cross-Site scripting attacks are easy to execute, but difficult to detect and prevent.[1,4]

E. System Architecture

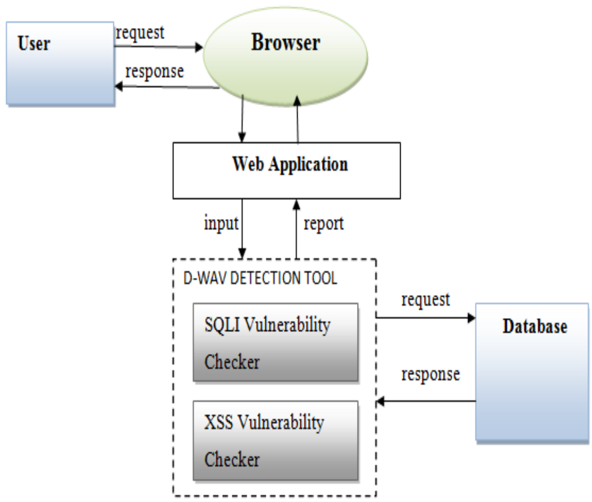

Figure1. Proposed System Architecture

III. WORKING

A. DETECTION OF ATTACKS

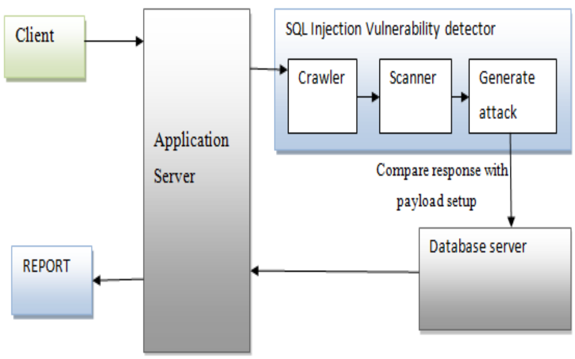

Figure2. SQL Injection and XSS Detection Architecture

Our tool checks for SQL injection vulnerabilities in the following steps.

1) Crawling the whole web application

For finding the input points we first explore the whole web application. In order to examine the entire web application it is designed in the form of a tree. Figure3 shows the tree structure of web application where a.php is the home or index page and the other pages are child nodes. After construction of the tree the pages are visited all the links are displayed in the working log.

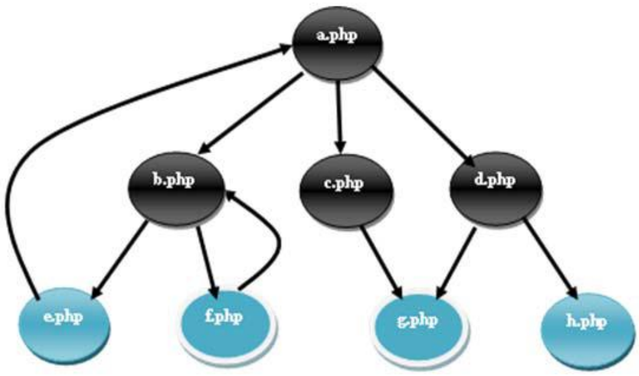

Figure3. Tree structure of web application[1]

2) Scanning the whole web application

The tool examines the URL of each visited and tries to identify the input points. If the page accepts user inputs then it is tagged as a vulnerable point. For example, if we get an URL like http://xyz.ac.in/departments/cse/csecourses.html then we can say that, in this page we do not have any vulnerable points. But if the URL is like http://xyz.site.com/product.php?product_id=10 then we can say product_id takes part in generation of a SQL query which may be of the form: SELECT * FROM product WHERE product_id=10. In this query product_id is the parameter and value is 10. Also if the URL is http://xyz.site.com/product.php?product_id=10 then we can say product_id takes part in generation of a XSS attack using script tag. The parameter element is always fixed but an attacker can freely alter the value element. Thus this URL has a vulnerable point.

3) Generate Attack and Report

The final step is to inject the attack codes at the vulnerable spots identified in the last step and report the outcome of the attack

3.1) Payload Setup

In this phase the attack payload is created based on the prevalent SQLI attacks and XSS attack. For generating the payload we created a list of the common SQL Injections and XSS attack which are used by attackers. The response of an attack will differ depending on the underlying database.

3.2) Generating Attack

In this phase we generate the attack by concatenating the attack payload with the original query URL of the web application, and make request of this specially crafted attack URL.

By sending different specially crafted attack request the proposed scanner checks if SQL injection and XSS vulnerabilities lie in a web application or not. For checking vulnerability we have defined a payload setup in which we have stored the attacks pattern related to different injection attack. We generate the attack request by appending attack pattern with the URL. After putting the attack request our tool automatically checks the response if there exist any vulnerability or not. If any vulnerability is found in the content of the response page then we can say that vulnerability exists in the input point of this page.

3.3) Generating Report

If any vulnerability exists in the web application, then a pdf report is generated indicating the date and time, the domain name and the SQL and XSS attacks found .

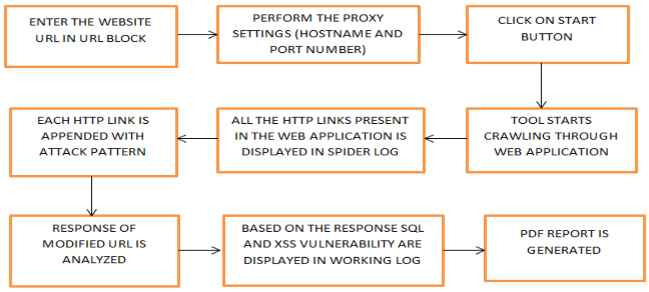

Figure4. Flowchart for SQL Injection and XSS Detection

IV. SYSTEM ANALYSIS

Functional Requirements

The D-WAV detection tool should be able to detect SQL Injection and Cross-site scripting attacks in any given web application. This tool will help the user to perform secure online transactions through web application.

V. INTENDED USE

A. EXPECTATIONS

The D-WAV detection tool should provide security to Web Applications by detecting the vulnerabilities such as SQL Injection and Cross-site scripting attacks. The tool is useful in securing the database of the web application.

B. CHALLENGES

The D-WAV detection tool should be designed in such a way that even a common man should be able to understand the process without any ambiguity. More and more web based enterprise applications deal with sensitive financial and medical data, which, if compromised, in addition to downtime can mean millions of dollars in damages. The hackers should not be able to cut short the performance of these web applications in any manner. Detecting the vulnerabilities in the web applications is the major challenge.

SYSTEM REQUIREMENTS

The software requirements identified for developing the application are JDK 1.7, Eclipse and APACHE TOMCAT as server.

A. SOFTWARE REQUIREMENTS

1. ECLIPSE

In computer programming, Eclipse is an integrated development environment (IDE). It contains a base workspace an extensible plug-in system for customizing the environment. Written mostly in Java, Eclipse can be used to develop applications. Eclipse is a multi-language software development environment comprising a workspace and an extensible plug-in system. It is written mostly in Java. It can be used to develop applications in Java and, by means of various plug-in, other programming languages.[5]

2. *WINDOWS OS*

Windows OS, computer operating system (OS) developed by Microsoft Corporation to run personal computers (PCs). Featuring the first graphical user interface (GUI) for IBM-compatible PCs, the Windows OS soon dominated the PC market. Approximately 90 percent of PCs run some version of Windows. The first version of Windows, released in 1985, was simply a GUI offered as an extension of Microsoft's existing disk operating system, or MS-DOS. [6]

3. *APACHE TOMCAT*

Apache Tomcat is an open source software implementation of the Java Servlet and Java Server Pages technologies. The Java Servlet and Java Server Pages specifications are developed under the Java Community Process. Apache Tomcat is developed in an open and participatory environment and released under the Apache License version 2. Apache Tomcat is intended to be a collaboration of the best-of-breed developers from around the world.[5]

B. HARDWARE REQUIREMENTS

- Intel Pentium 4 processor or higher.
- Minimum RAM of 512mb.
- Free disk space of 16GB or more.
- 1024 x 768 resolution monitor.

C. TECHNOLOGY

JAVA

Java was originally called OAK, and was designed for handheld devices and set-top boxes Java is an object-oriented language similar to C++, but simplified to eliminate language features that cause common programming errors Java is a general purpose programming language with a number of features that make the language well suited for use on the World Wide Web. Small Java applications are called Java applets and can be downloaded from a Web server and run on your computer by a Java-compatible Web browser, such as Netscape Navigator or Microsoft Internet Explorer.[7]

D. DESIGN AND IMPLEMENTATION ISSUES

One major issue in the design and implementation was that if the attack pattern stored in our code is not correct then tool will not be able to detect the attack successfully. Also if the hacker performs any new type of attack of which the pattern is not stored in our code then the tool will not be able to detect that attack.

VI. RESULT

A. SQL INJECTION DETECTION

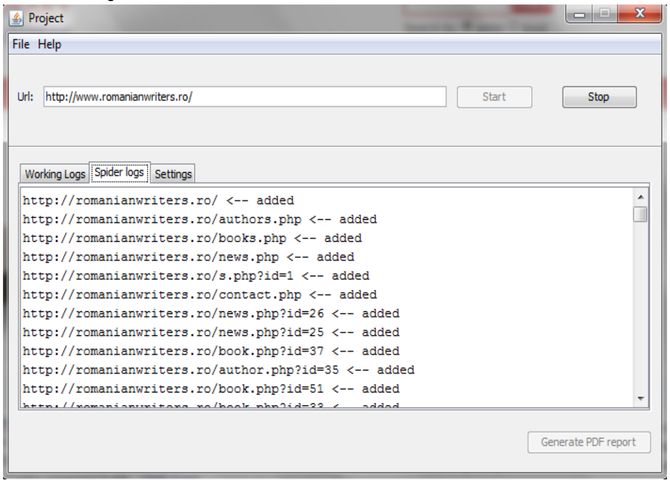

Figure5. Crawler of SQL Injection detection

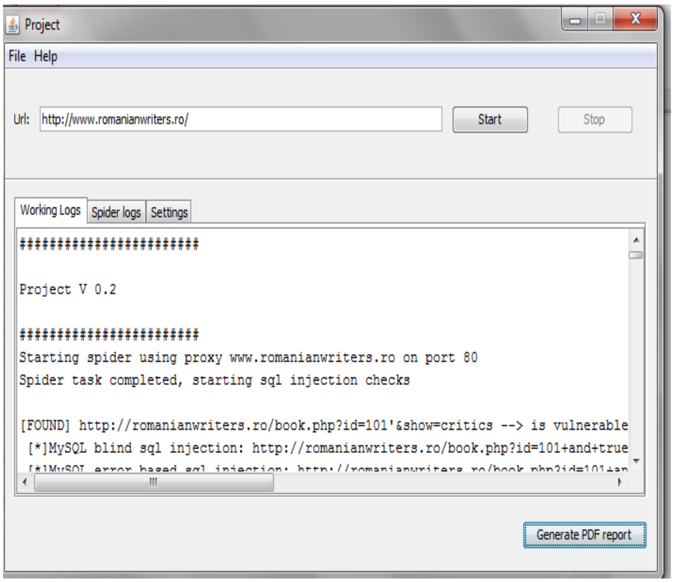

Figure6. Working log of SQL Injection detection

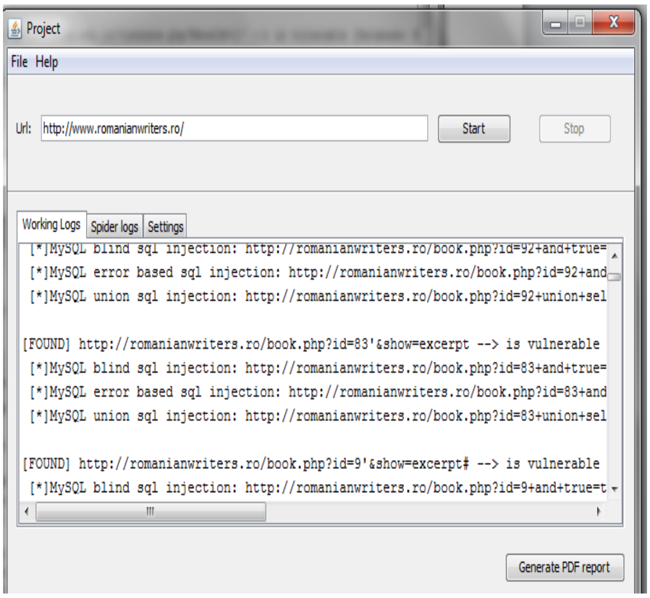

Figure7. Vulnerabilities displayed in the working log

B. *CROSS-SITE SCRIPTING DETECTION*

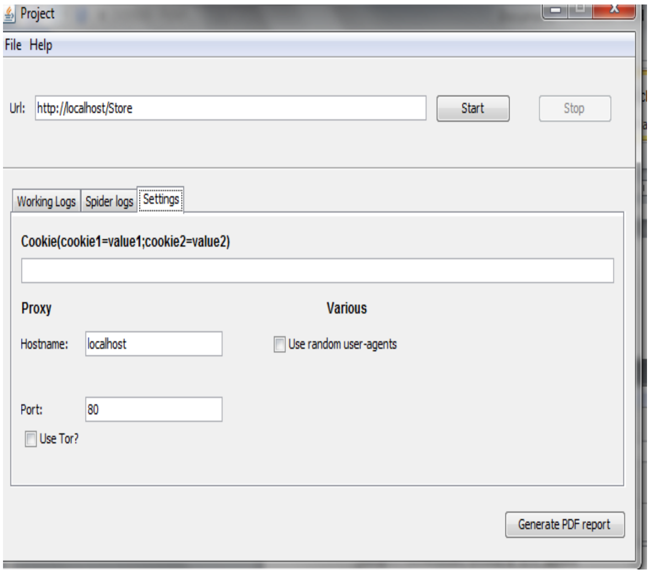

Figure8. Proxy Setting

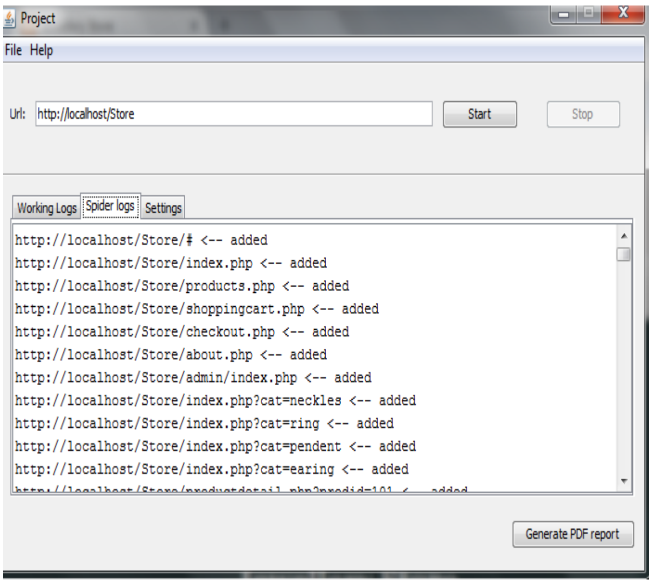

Figure9. Crawler of XSS detection

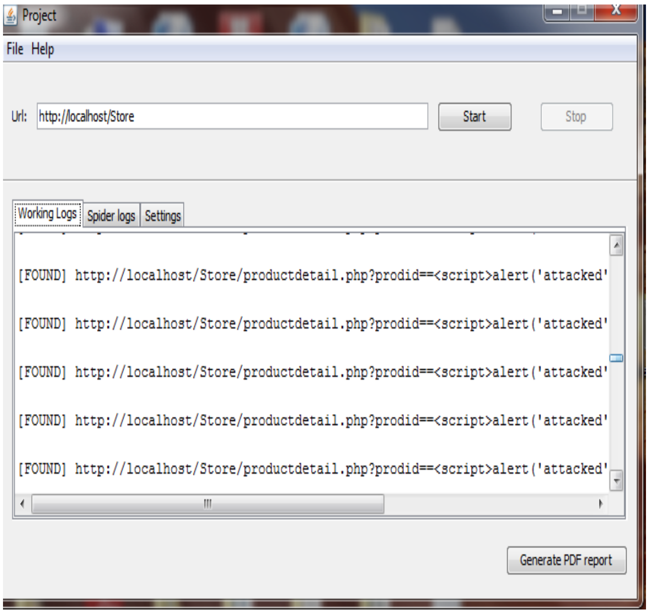

Figure10. Working log of XSS detection

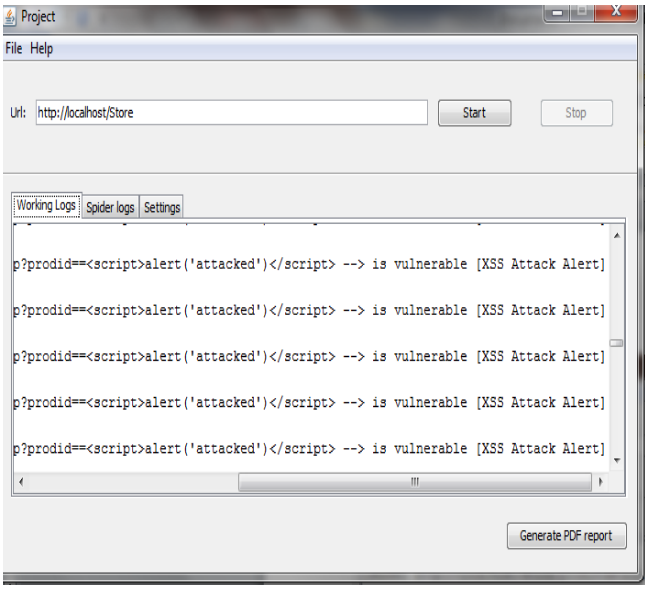

Figure11. Working log of XSS detection

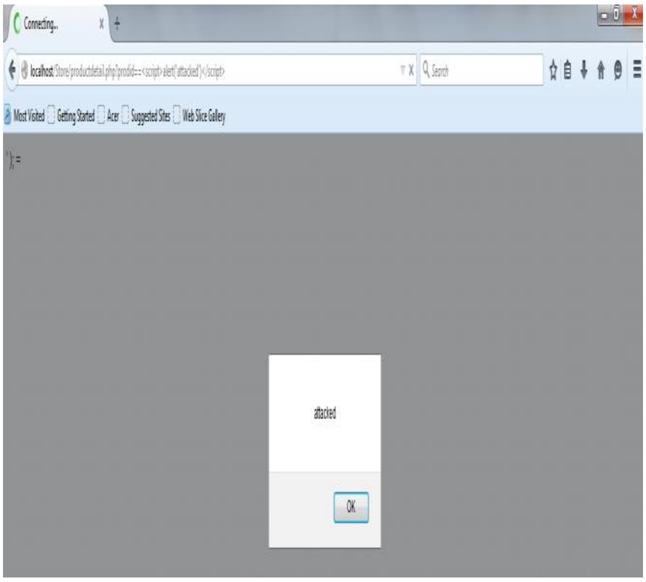

Figure12. Alert Box displayed on generation of XSS attack

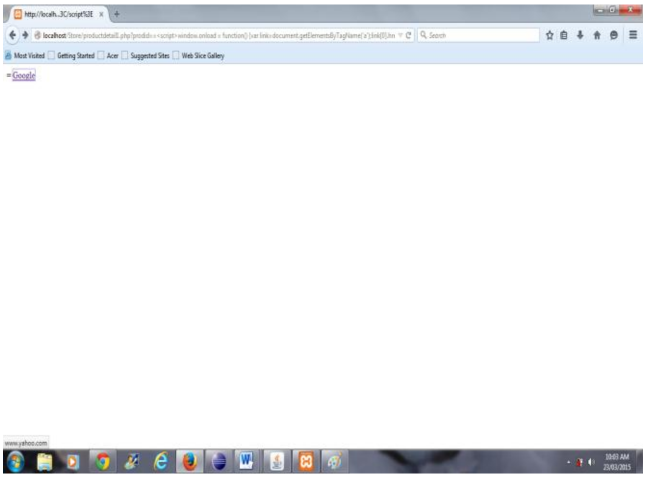

Figure13. Web page redirected to another link on generation of XSS attack

Project Report

Report generated on: Mon Feb 23 22:27:15 IST 2015

Domain: romanianwriters.ro

SQL Errors founds:

Url	type of vulnerability
http://romanianwriters.ro/book.php?id=101'&show=critics	Found SQL error
http://romanianwriters.ro/book.php?id=83'&show=excerpt#top	Found SQL error
http://romanianwriters.ro/book.php?id=34'&show=excerpt#	Found SQL error
http://romanianwriters.ro/book.php?id=31'&show=excerpt#	Found SQL error
http://romanianwriters.ro/book.php?id=67'&show=excerpt#top	Found SQL error

Figure11. Pdf Report generated when the vulnerabilities are detected in the web application

VII. FUTURE ENHANCEMENTS

Due to the issues with the chosen technology and the development process, the development team has decided to include more attack patterns for the better performance.

VIII. CONCLUSION

The objective of the project is to detect SQL Injection and Cross-site Scripting attacks by checking the URL of the web application and thus provides security to web application by detecting the attacks.

ACKNOWLEDGEMENT

The author wishes to thank our guide for helping out to come up with the initial idea and guiding us to proceed further in the project. Also would like our project HOD for giving us the opportunity to do this project.

REFERENCES

[1] Sangita Roy, Avinash Kumar Singh and Ashok Singh Sairam, senior member IACSIT "Detecting and Defeating SQL Injection Attacks "International Journal of Information and Electronics Engineering, Vol. 1 , No. 1 , July 2011".

[2] OWASP (Open Web Application Security Project) https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project visited on August 2014.

[3]https://www.owasp.org/index.php/Testing_for_SQL_Injection_(OTG-INPVAL-005) visited on November 2014

[4] Sudhir S. Dhekane1, Prof. V. B.Gaikwad2," XSS Detection in Web Request and Server Response" International Journal of Emerging Technology and Advanced Engineering Website: www.ijetae.com (ISSN 2250-2459,ISO 9001:2008Certified Journal, Volume 4, Issue 4, April 2014)

[5] http://en.wikipedia.org/wiki/Eclipse_(software) visited on September 2014.

[6]http://www.britannica.com/EBchecked/topic/645197/Windows-OS visited on September 2014.

[7] http://www.webopedia.om/TERM/J/Java.html visited on September 2014.

[8] A. Duraisamy, M.Sathiyamoorthy, S.Chandrasekar, " A Server Side Solution for Protection of Web Applications from Cross-Site Scripting Attacks." International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278 - 3075, Volume-2, Issue-4, March 2013

[9] Atefeh Tajpour, Maslin Masrom, Mohammad Zaman Heydari, Suhaimi Ibrahim,"Evaluation of SQL Injection Detection and Prevention Techniques,"2nd International Conference on Computational Intelligence,Communication Systems and Networks, Liverpool, UnitedKingdom. 216-221.