ISSN : 2278 – 1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 5, July 2012

An Approach of Preventing Code Injection Attack in Web Environment

Nikita Patel[1], Prof. Shivshakti Shrivastava[2], Prof Hitesh Gupta[3]

Patel College of Science & Technology, Bhopal, India[1,2,3]

nikitabist[1]@ gmail.com

ABSTRACT— ***Nearly every web application is developed in a way that the demand of specific data is completed through user input. When dynamic SQL query is used there is a possibility that user may insert malicious code as input in order to have access of sensitive information or unauthorized access of the database [7]. We have developed a new mechanism to prevent SQL injection attack, in this method we have stored some exceptional information in the database which cannot be used in normal case, our application look for such exceptional information, if it is found in accessed data then request is discarded by the web server.***

Keywords: SQL Injection, Malicious Code, Attacker, Unauthorized Access

I. INTRODUCTION

Today web applications are significant part of our life. The end users communicate with web application by using web browser. The security of web application is major concern for web developers, security vulnerability are frequently exploited by attacker or hacker. Vulnerabilities may provide a way to an unauthorized person to gain access to essential information, use resources inappropriately, interrupt business or commit fraud [1]. With the help of attack(s) attacker can gain access of whole web site or web server. Attacker also can access associated database which may have sensitive information of an organization. Email service, shopping portals and social websites are commonly used web applications. The hackers examine a web application and understand its design, identify any weak points, and use these weaknesses to exploit the application. Source code comments, Error messages, HTML source code view can also assist to hacker for understanding of web application. There are main three components which are shown in figure 1.

This paper is organized into ten sections including this one. The second section gives an idea of web application vulnerability. Third and fourth section shows the code injection and SQL injection attack. Section five considers the existing technique of the SQL injection attack. Section six shows the proposed architecture. Section seven has all the detail regarding implementation of defence mechanism, algorithm, flow chart and results. Finally the paper concludes in section eight.

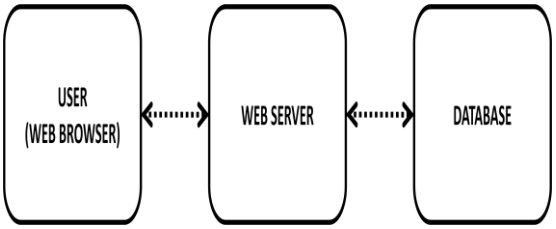


Figure 1 Web Application Components

II. WEB APPLICATION VULNERABILITY

Vulnerability is a gap or a weak point in the application or software, which can be occurred during Software Development Life Cycle (SDLC), which permits an attacker to exploit web application [3].

There are numerous weaknesses in the web application, which can be exploited to complete a malicious mission. Evaluations of web application security are continuously being researched, both by attackers and by security professionals. These weaknesses affect all active web applications at the same time as others are reliant on specific application. The development and enhancement of web technologies also introduces new exploits which compromise with insightful information and provide access to unauthorized persons [4].

These are the common web application vulnerabilities.

- SQL INJECTION
- CODE EXECUTION
- COOKIE MANIPULATION
- HTTP RESPONSE SPLITTING

ISSN : 2278 – 1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 5, July 2012

- CROSS SITE SCRIPTING

III. CODE INJECTION ATTACK

Code Injection is an expression used when malevolent code is inserted into a web application from remote location, for example input field which is provided for taking input from the user. The lack of accurate input or input without sanitization lets an attacker to inject malicious and injected code executes as a part of application. The end result of code injection attacks either leak sensitive information, or an undesirable operation. Attack can be performed within software, web application etc in which the weakness is present. Weaknesses or vulnerabilities can be used by an attacker to take advantage of the web applications or to gain unauthorized access of information, denial of services, or perform incorrect operations. HTML code Injection, XSS, SQL Injection, HTTP Request splitting and XML Poisoning Attack are the examples of the code injection attack [2].

IV. SQL INJECTION ATTACK

SQL injection is an attack method by which attacker can inject malicious query for exploiting the web application; By SQL injection attack, attacker might gain unauthorized access to a database or to take back sensitive information directly from the database. Attacker exploits SQL injection vulnerabilities distantly without help of any special tool. SQL injection attacks are undemanding in nature; an attacker just goes with malicious query as input to an application for accessing private information [2].

4.1. EXPLANATION OF SQL INJECTION ATTACK

Suppose we have a web application, using this web application user can see his/her result by submitting roll number in the text field.

Submitted roll number is used for accessing particular result from database. Using this roll number programmer make dynamic query it may be like.

*SELECT * FROM student WHERE srollnum = 'Given_Roll_Number';*

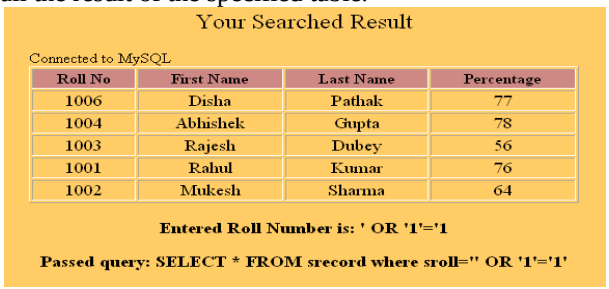
The above query is a standard query for accessing data from student table where

student = Table Name in the database
srollnum = Attribute name of student table
Given_Roll_Number = Roll number submitted by result viewer or student

When this query is executed it returns result. The data, filling by the remote user, is appears to be in the WHERE clause. Let's see what happens if we submit this malicious query [5].

*SELECT * FROM student WHERE srollnum = ' ' OR '1'='1';*

In reality the web applications do not focus on query - simply making a dynamic query - our use of quotes has turned a single-component WHERE clause into a two-component one, and the '1'='1' part is always true no matter what the first clause is, that way this malicious query returns all the result of the specified table.


Your Searched Result

Connected to MySQL

Roll No	First Name	Last Name	Percentage
1006	Disha	Pathak	77
1004	Abhishek	Gupta	78
1003	Rajesh	Dubey	56
1001	Rahul	Kumar	76
1002	Mukesh	Sharma	64

Entered Roll Number is: ' OR '1'='1

Passed query: SELECT * FROM srecord where sroll='' OR '1'='1'

Figure 2 Result of Malicious Code

V. EXISTING TECHNIQUES OF SQL INJECTION ATTACK PREVENTION

There are many other techniques for prevention of SQL injection [8] like

- Using Client Side Scripting Language like JavaScript
- Parameterized Query
- Stored Procedure
- Regular Expression to Discard Input String

VI. PROPOSED DEFENCE MECHANISM

Web Application executes in two tier architecture, which consist a web server for handling end user requests, application server which executes application programs written in any programming language and Database server for storing data. The defence mechanism is focused on the contents which are sent by the database to application server as a response of requested data. Now our application code examines for exceptional content, which are inserted in the database for prevention of such type of attacks, if such exceptional contents are found in responded data then the application server deny for requested data. If such type of request comes again and again, in that we can block that particular IP address for some period of time. Proposed architecture shows the defence mechanism in figure 3.

ISSN : 2278 – 1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 5, July 2012

VII. PROPOSED ARCHITECTURE

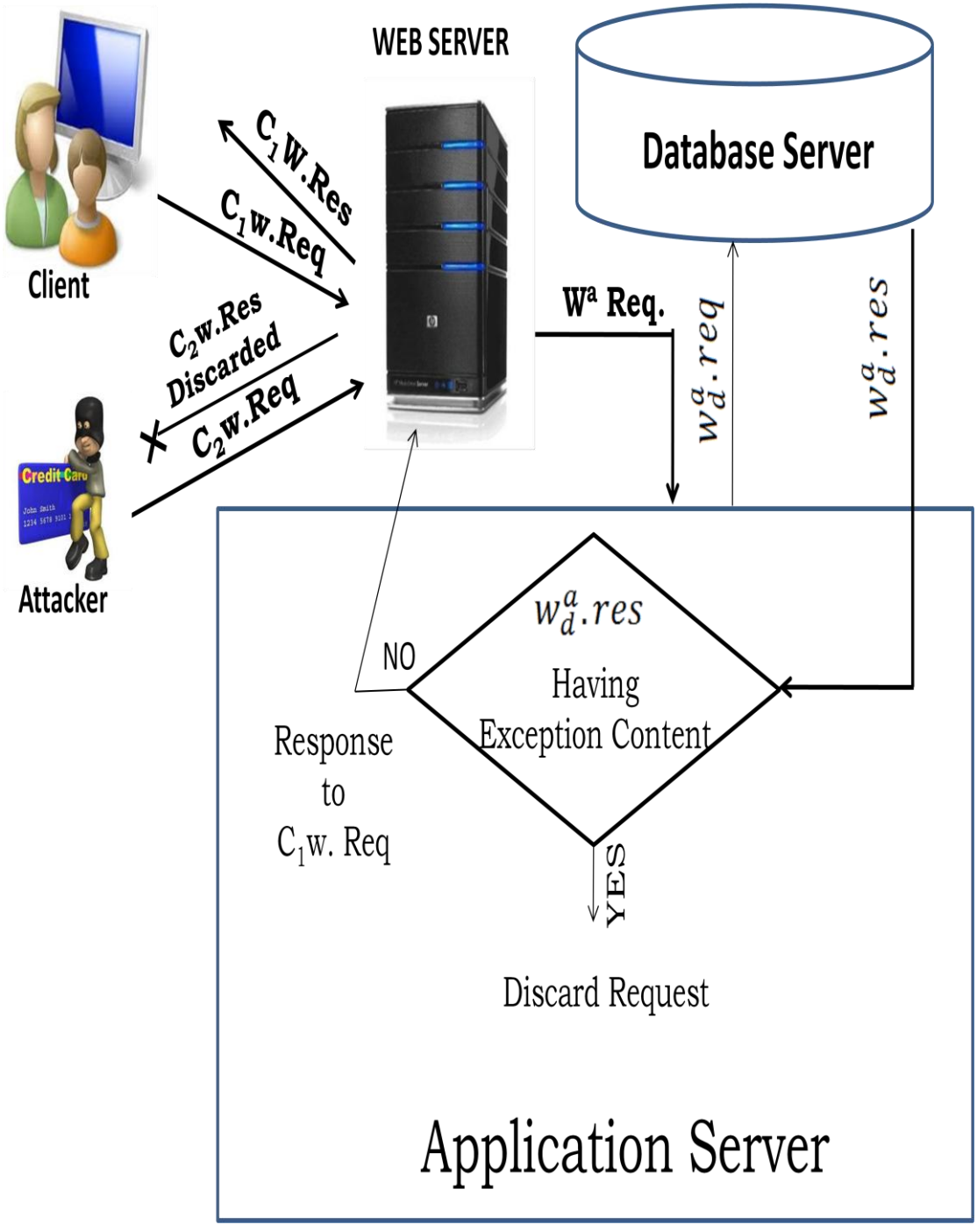


Figure 3 Proposed Architecture

ISSN : 2278 – 1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 5, July 2012

GO TO STEP 7

7.1. PROPOSED ALGORITHM

Assumptions

$$C_i = Client\ i$$

W.req = Web Request

$$W_s = Web\ Server$$

$$A_s = Application\ Server$$

$$D_s = Database\ Server$$

$$E_c = Exceptional\ Content$$

STEP 1:

Any client C_i sends a request C_{iw} for a web service

STEP2:

Web server processes and forward this request to the application server

$W_s(C_{iw}.req) \rightarrow C_{iw}^a.req$

STEP 3:

Application server forwards this $C_{iw}^a.req$ to database server.

$A_s(C_{iwd}^a.req) \rightarrow C_{iwd}^a\ req$

STEP 4:

Database server processes and sends respond back to application server.

$D_s(C_{iwd}^a\ req) \rightarrow C_{iw}^a\ res$ (Response)

STEP 5:

Application Server Check for exceptional data

If $\exists\ E_c(C_{iw}^a\ res)$

GO TO STEP 6

Else

STEP 6:

Requested Service will be discarded

STEP 7:

Requested Service will be granted

7.2. FLOW CHART

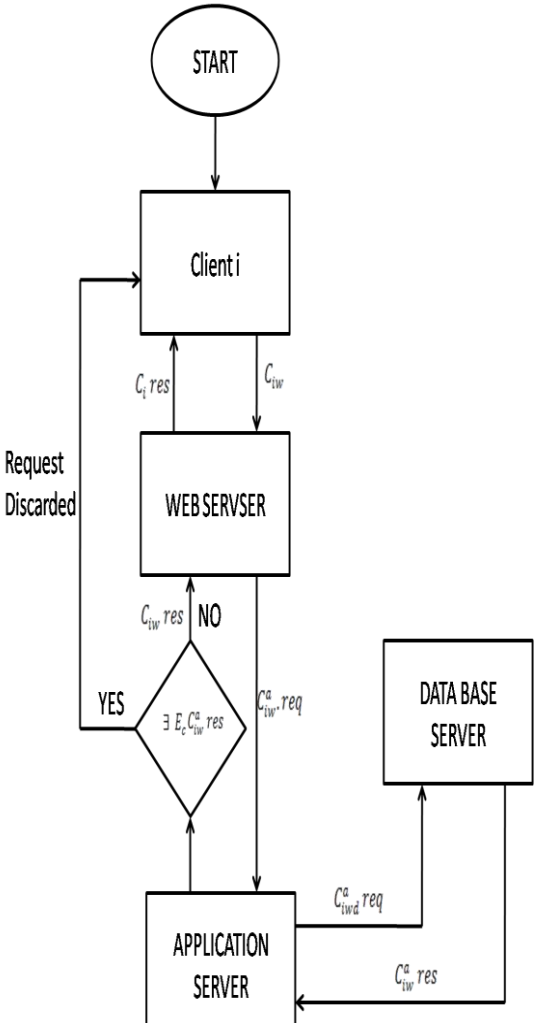


Figure 4 Flow Chart

ISSN : 2278 – 1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 5, July 2012

7.3. RESULTS

Figure 5 & 6 shows the output result of the proposed method. As the motive was stopped sql injection attack has fulfill at the end of this work.

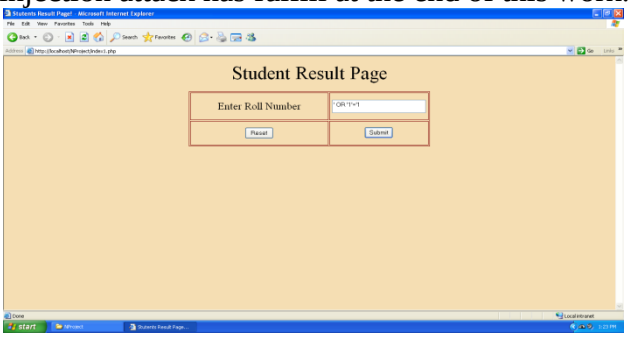


Figure 5 Input Window

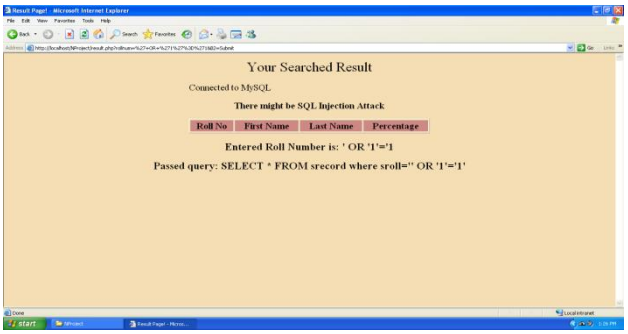


Figure 6 Output Result after Defence Mechanism

VIII. CONCLUSION

The version of this template is V2. Most of the formatting instructions in this document have been compiled by Causal Productions from the IEEE LaTeX style files. Causal Productions offers both A4 templates and US Letter templates for LaTeX and Microsoft Word. The LaTeX templates depend on the official IEEEtran.cls and IEEEtran.bst files, whereas the Microsoft Word templates are self-contained. Causal Productions has used its best efforts to ensure that the templates have the same appearance.

IX. ACKNOWLEDGMENT

The presented research would not have been possible without our college, PCST, Bhopal. We wish to express our appreciation to all the people who helped turn the World-Wide Web into the useful and popular distributed hypertext and providing information as it is anywhere. We also wish to pay thanks the anonymous reviewers for their valuable suggestions, who helped in improving our work.

REFERENCES

[1] "Common Web Application Vulnerabilities", Available at www.computerworld.com, Accessed on 12 Nov 2011.
[2] Nikita Patel, Fahim Mohammed, Santosh Soni, *"SQL Injection Attacks: Techniques and Protection Mechanisms"*, IJCSE, JAN, 2011.
[3] *"Vulnerability"*, Available at www.owasp.org, Accessed on 05 Nov 2011.
[4] *"Vulnerability"*, Available at www.acunetix.com Accessed on 05 Nov 2011.
[5] *"SQL Injection Attack"*, Available at www.unixwiz.net, Accessed on 21 Nov 2011.
[6] *"Prevention Technique of SQL Injection Attack"*, Available at www.codeproject.com, Accessed on 22 Nov 2011].
[7] Jin-Cherng, Lin and, Jan-Min Chen, "The Automatic Defense Mechanism for Malicious Injection Attack", IEEE, 2007
[8] Endler & David, "The Evolution of Cross-Site scripting Attacks", IDEFENSE Labs, 2002.
[9] R. Ezumalai, G. Aghila, "Combinatorial Approach for Preventing SQL Injection Attacks", IEEE 2009, pp 1212-1217.
[10] Allen Pomeroy and Qing Tan, "Effective SQL Injection Attack Reconstruction Using Network Recording" IEEE 2011, pp 552-556
[11] Li Shan Dong, Xiaorui & RaoHong , "An Adaptive Method Preventing Database from SQL Injection Attacks", IEEE2010, pp 352-355.