Enhanced Intrusion Detection System for Input Validation Attacks in Web Application

Puspendra Kumar[1], R. K. Pateriya[2]

[1] M Tech Scholar ,Computer Science & Engineering Department,
Maulana Azad National Institute of Technology
Bhopal, 462051, India

[2] Associate Professor, Computer Science & Engineering Department,
Maulana Azad National Institute of Technology
Bhopal, 462051, India

Abstract

Internet continues to expand exponentially and access to the Internet become more prevalent in our daily life but at the same time web application are becoming most attractive targets for hacker and cyber criminals. This paper presents an enhanced intrusion detection system approach for detecting input validation attacks in the web application. The existing IDS for Input validation attacks are language dependent. The proposed IDS is language independent i.e. it works for any web application developed with the aid of java, php, dot net etc. In addition the proposed system detects directory traversal attacks, command injection attacks, cross site scripting attacks and SQL injection attacks, those were not detected in the existing IDS. This is an automatic technique for detection vulnerabilities over the internet. Our technique is based on the web application parameter which is in form of POST and GET which has generalized structure and values. This technique reduces analysis time of input validation attacks.

Keywords: SQL Injection attacks, XSS attacks, directory traversal attacks, GET and POST data, Detection.

1. Introduction

Web Application are most widely used for providing service to the user like online shopping, online reservation, and many more application which is designed in perspective of user. So the web application is popular attacks target due to time and financial constraints, limited knowledge of the programming, limited knowledge security awareness, misconfiguration that is meant lack of awareness of the security configuration deployment on the part of the programmer. With the aid of the input validation attacks attacker can steal the confidential data which decrease the market values of the organization. Web applications generally use TCP port for the communication with server. This communication is not protected by the IVAs [1].

Current Intrusion detection systems are designed in such a way that detects SQL injection attacks, XSS attacks but they do not detect the directory traversal attacks, and command injection attacks. Such IDS are designed language specific for example IDS for PHP based web application, IDS for JAVA based web application [2, 3, 4]. The proposed enhanced IDS approach detect SQL Injection attacks, XSS attacks, Directory Traversal attacks and command injection, and is not programming language specific. This IDS approach require only window environment for detecting IVA over the internet. It analyze web request data to detect that if any type of IVA exists. Then the detection system automatically generate the report against input validation attacks and send it to the server owner.

The enhanced IDS approach can be more effective for finding out any type of Input validation attacks and with the aid of this approach server administrator can take effective action against these attacks. So in this way this approach reduces the analysis time and also increase the efficiency of the system.

Rest of the paper is organized as follows: Section 2 describes the input validation attacks and section 3 describes the related work. Proposed enhanced IDS approach is given in section 4 this section also discusses how to analyze the raw data. Section 5 describes the comparison with existing IDS. Finally conclusion is given in section 6.

2. Input Validation Attacks

The Input Validation Attacks (IVAs) attempt to submit data which the web application does not expect to receive, that causes very serious consequences like session hijack, SQL poisoning, source code disclosure, directory browsing etc. Input validation is a security issue if an attacker

discovers that the application makes unfounded assumptions about the type, length, format, or range of input data. The attacker can then supply carefully crafted input that compromises the application. When network and host level entry points are fully secured; the public interfaces exposed by the application become the only source of attack. The input to the application is a means to both test the system and a way to execute code on an attacker's behalf. If the applications blindly trust input. It may be susceptible to the following:

2.1 Buffer Overflows Attacks

Buffer overflow vulnerabilities can lead to denial of service attacks or code injection. A denial of service attack causes a process crash. Code injection alters the program execution address to run an attacker's injected code.

2.2 Cross-Site Scripting Attacks

An XSS attack can cause arbitrary code to run in a user's browser while the browser is connected to a trusted Web site. The attack targets the application's users and not the application itself, but it uses the application as the vehicle for the attack. Because the script code is downloaded by the browser from a trusted site, the browser has no way of knowing that the code is not legitimate. Internet Explorer security zones provide no defense. Since the attacker's code has access to the cookies associated with the trusted site and are stored on the user's local computer, a user's authentication cookies are typically the target of attack.
Example of Cross-Site Scripting:
To initiate the attack, the attacker must convince the user to click on a carefully crafted hyperlink, for example, by embedding a link in an email sent to the user or by adding a malicious link to a newsgroup posting. The link points to a vulnerable page in the application that echoes the invalidated input back to the browser in the HTML output stream. For example, consider the following two links.
Here is a legitimate link:
www.webapplication.com/logon.aspx?username=puspendra
Here is a malicious link:
www.webapplication.com/logon.aspx?username=<script>alert ('hacker code') </script>
If the Web application takes the query string, fails to properly validate it, and then returns it to the browser, the script code executes in the browser. The preceding example displays a harmless pop-up message. With the appropriate script, the attacker can easily extract the user's authentication cookie, post it to his site, and subsequently make a request to the target Web site as the authenticated user.

2.3 SQL Injection Attacks

A SQL injection attack exploits vulnerabilities in input validation to run arbitrary commands in the database. It can occur when the application uses input to construct dynamic SQL statements to access the database. It can also occur if code uses stored procedures that are passed strings that contain unfiltered user input. Using the SQL injection attack, the attacker can execute arbitrary commands in the database. The issue is enhance if the application uses an over-privileged account to connect to the database. In this instance it is possible to use the database server to run operating system commands and potentially compromise other servers, in addition to being able to retrieve, manipulate, and destroy data.

2.4 Canonicalization

Different forms of input that resolve to the same standard name (the canonical name), is referred to as canonicalization. Code is particularly susceptible to canonicalization issues if it makes security decisions based on the name of a resource that is passed to the program as input. Files, paths, and URLs are resource types that are vulnerable to canonicalization because in each case there are many different ways to represent the same name. File names are also problematic.

3. Related Work

3.1 ARDILLA Tool[2]

This tool is developed by Adam Kie˙zun, Philip J. Guo, Karthick Jayaraman, Michael D. Ernst. This tool is useful for identifying the SQL Injection attacks and XSS vulnerabilities. This technique works on unmodified existing code, generate concrete input that expose vulnerabilities and operate before software is deployed. ARDILLA is an automated tool for creating attacks. It is white box testing tool means that it requires source code of the application. It is based on the input generation, taint propagation and input mutation to find variants of an execution that exploit vulnerability.

3.2 Protect Web Application using Positive Tainting and Syntax-Aware Evaluation[3]

This approach is proposed by William G.J. Halfond, Alessandro Orso and Panagiotis Manolios. This approach uses four term to detect SQLIA Positive tainting, Accurate and efficient taint propagation, Syntax-aware evaluation of queries string and Minimal deployment requirement.

3.3 VIPER for Detecting SQL Injection Attacks[4]

In this techniques SQL Injection attacks detected by using heuristic based approach. It basically performs panetration testing of the web application .This approach analyzes the web application for detrmininng hyperlinks structure and input supplied from the user and gives error message, if any type of SQL injection occurres.

3.4 Runtime Monitoring Technique[5]

AMNESIA technique is used for detecting SQL Injection attacks over the web application. These technique workes on both static approach and runtime monitoring. It detects injected query before executed on the database server using model based approach.This approach have two part static part which automatically builds a legel queries using program analysis on the other hand in dynamic part it dynamically generates the queries against statically build queries using runtime monitoring. If queries violate the approach then this approach perevents the execution of the queries on the database server. This technique has four steps for preventing injection Identify the hotspot, Build SQL-query models, Instrument application, Runtime monitoring.

3.5 Detection by Feature of Single Character[6]

This techniques used to sigmoid function for detecting SQL injection attacks. They proposed detection algorithm of SQL Injection Attack based on single character. When the SQL character string is the SQL Injection, it call an attack character string. With the aid of this approach minimizes the predictive error in SQL Injection attack detection.

3.6 Obfuscation-based Analysis of SQL Injection Attacks[7]

This approach is proposed by Raju Halder and Agostino Cortesi. They implemented combined structure of static and dynamic analysis which is based on the obfuscation and de-obfuscation of SQLcommands SQL Injection attacks can be easily detected because dynamic verification is carried out on the obfuscated queries , at atomic formula level only those atomic formulas are tagged as vulnerable. And this approach finds the root cause of the SQL Injection attacks in dynamic query generation.

4. Enhanced IDS Approach

This section describe the enhanced IDS for Input Validation Attacks in web application. This proposed approach will perform on the Application layer of the OSI model. So that with the aid of application layer we can obtain POST and GET data over the network communication with server. C# language using socket programming [8] will be used to develop the proposed IDS. This approach will work in six steps for performing detection of input validation attacks (IVAs) as shown in following block diagram.

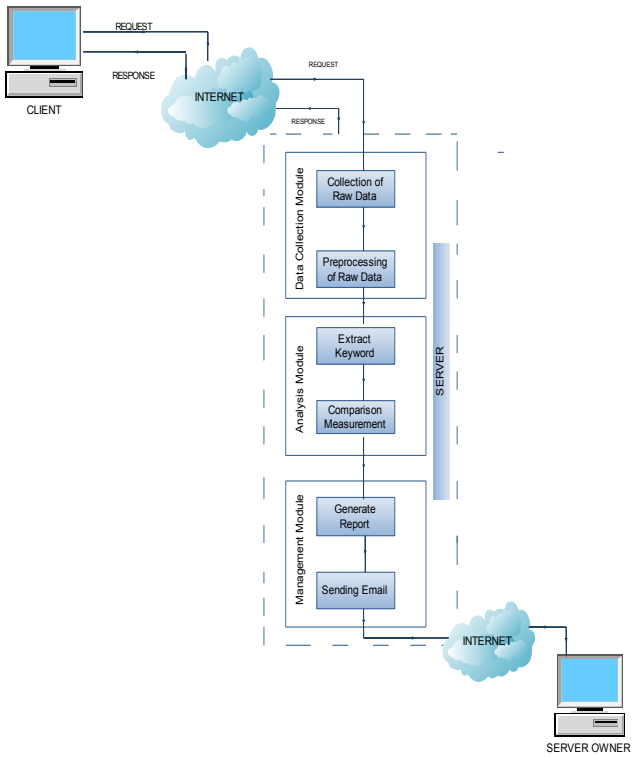


Fig. 1 Block Diagram of Proposed Enhanced IDS

4.1 Collection of Raw Data

In this step we collect the HTTP header data which contains GET and POST data which are used to pass the parameter value to the web server over the internet. The HTTP header data works as raw data for the proposed IDS. Raw data will be stored in text file. These file contains both request data and response data which is analyzed by proposed approach for finding out IVA attacks. The process of raw data collection is given stepwise in following DFD.

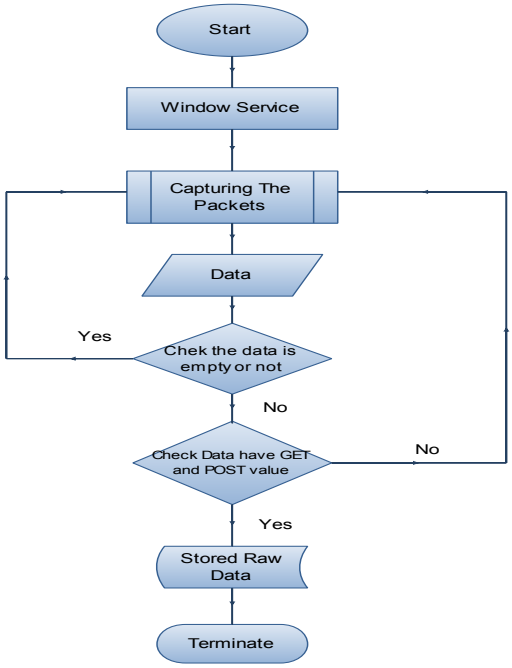
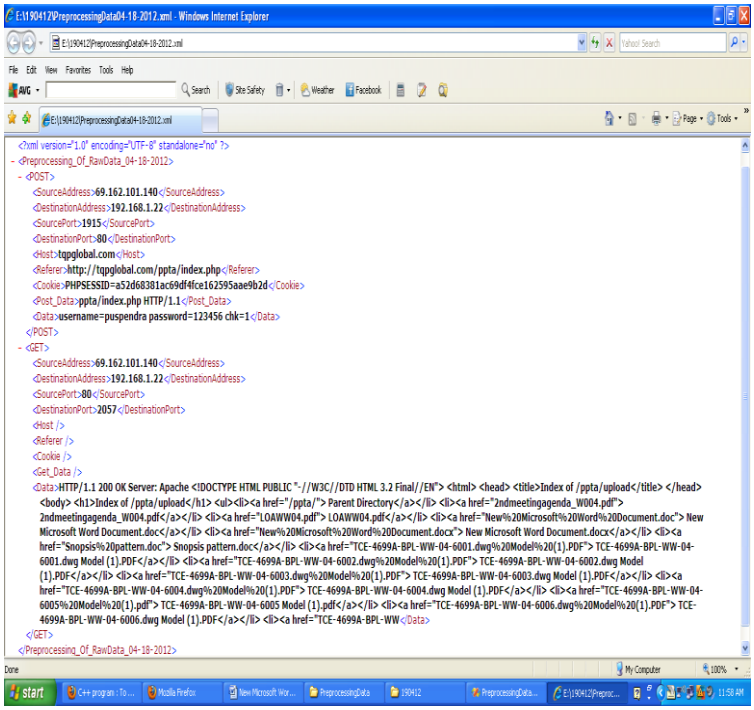


Fig. 2 DFD of Collection of Raw Data



4.2 Preprocessing of the Raw Data

The raw data collected in step 4.1 is preprocessed in this step the raw data is filtered to keep GET and POST data and to remove unnecessary information like http header details. These preprocessing data are generated in XML format. In this step we analyze the raw data and fetch useful data from raw text file. Following steps are used for preprocessing of the data.

Step 1: Input text file (.Text) containing raw data which is obtained in step1.
Step 2: Read file by line by line and match with HTTP header information like referrer, Host, and cookie etc. If match found then corresponding value will be stored in DataItem class. This task is performed until the entire HTTP header is not matched.
Step 3: After data stored in DataItem class then these add in the ArrayList collection.
Step 4: Repeat step 2 and step 3 until all of the data is read from text file.
Step 5: Add DataItem class in ArrayList collection. Now we need to fetch DataItem properties values like Host, Referrer and Cookie etc and assigned to the predefined the XML document.
Step 6: The generated output is in the form of XML format which contained the root node, child node.
Following screen shot shows the preprocessing of data in XML format.

4.3 Keyword Extraction

After preprocessing of the raw data we extract keywords from the xml file of the preprocessed data. In this step we extract GET and POST data from xml file or preprocessed file and perform analysis on extracted data. Following steps are used to Extract Keywords:
Step 1: Input the XML file which contains preprocessed data.
Step 2: Read the file according to Root node like POST and GET.
Step 3: If root node is POST then we extracts data of child node which will be source address, destination address, source port, destination port and its data, which is stored in Extracted DataItem collection class.
Step 4: If root node is GET then we extracts data of child node which will be source address, destination address, source port, destination port and its data, Get data to store in Extracted DataItem collection class. Otherwise go to step 6.
Step 5: Repeat step 3rd and step 4th until read operation in XML or preprocessing file is complete.
Step 6: End process.
The process of keyword extraction is given in following DFD.

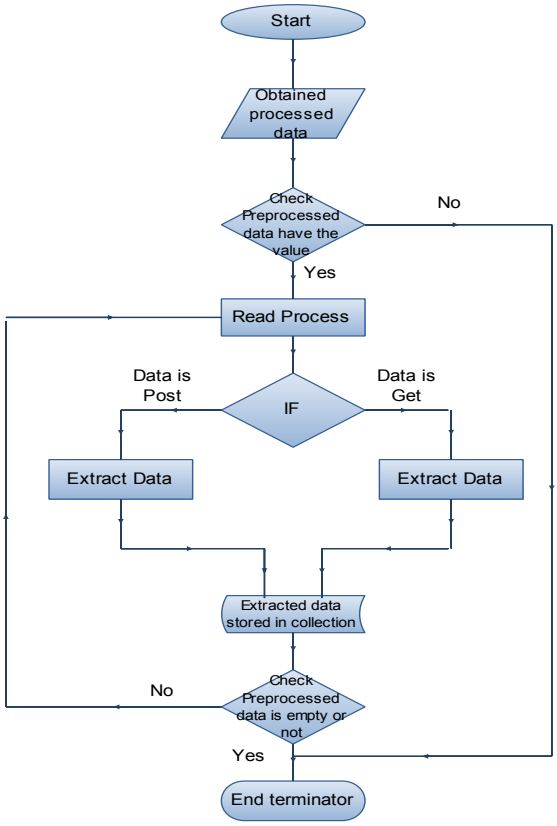


Fig. 3 DFD of Extracted Keyword

4.4 Comparison Measurement

In this step templates for SQL Injection attacks, cross site scripting attacks, command injection attacks directory traversal attacks will be created these templates will contain values corresponding to these attacks.

The data value of generated templates are compared with the data value of templates obtained in the step 4.3. If a match is found then these are marked as attacks like SQL Injection attacks, Cross site scripting attacks, command injection attacks and directory traversal attacks, and stored in the hard drive.

Following steps are used for the Comparison Measurement:

Step 1: Get the Extracted data which is found in step3.

Step 2: Perform matching process.

Step 3: Match data value with SQL Injection attacks template if match found then go to Step 7 otherwise go to next step.

Step 4: Match data value with Cross site scripting attacks template if match found then go to Step 7 otherwise go to next step.

Step 5: Match data value with Command Injection attacks template if match found then go to Step 7 otherwise go to next step.

Step 6: Match data value with Directory traversal attacks template if match found then go to Step 7 otherwise go to step 9.

Step 7: In this step data value is marked as an attack corresponding attacks template.

Step 8: In this step marked data value will stored in stored file.

Step 9: End process.

These steps clearly understood by data flow diagram of Comparison Measurement as shown in figure 5.

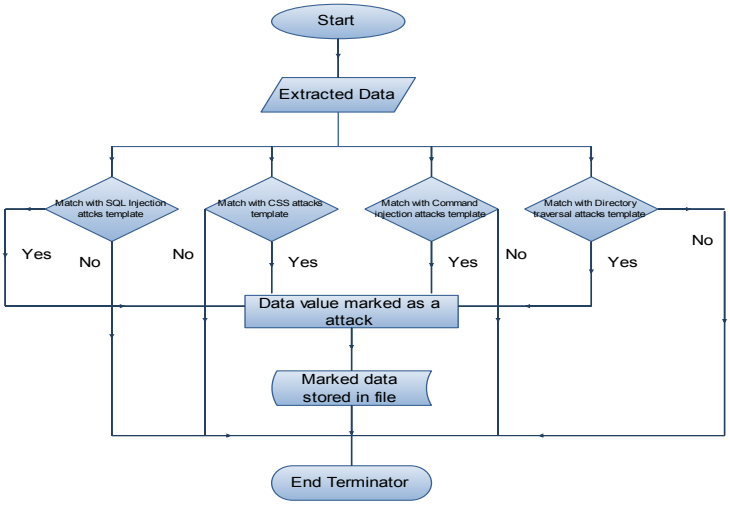
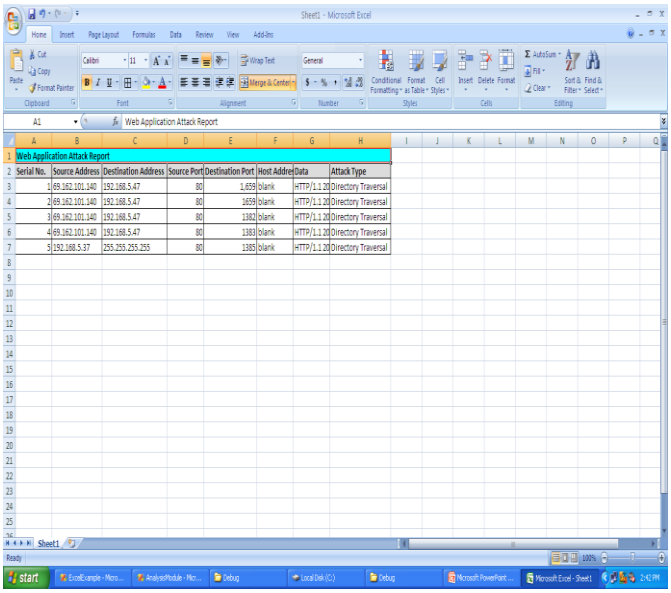


Fig. 4 DFD of Comparison Measurement

4.5 Report Generation

This step is for automatic report generation whenever any attacks is detected than automatic report is generated immediately in this step. This report has information about attacks like SQL Injection attacks, Command Injection Attacks, and Directory browsing attacks. The generated report also have the information about source address, destination address, source port, destination port, host, data and type of attacks performed on the data.

Following screen shots shows the attack report.



4.6 Sending Email

The generated report in step 4.5 is sent to server owner by email. With the aid of this report, server owner becomes aware with the online attacks on the server and take appropriate action to protect the server from web input validation attacks. This email sending is also automatic process.

5. Comparison with Existing IDS

This section shows that why proposed approach is better than previous IDS to detect input validation attacks in web application. Table 1 gives the comparative view of the existing IDS with proposed IDS.

Table 1: Comparison with existing IDS

Techniques	***Language***	**Attacks**
ARDILLA[2]	PHP based Web Application	Sql Injection , XSS attacks
WASP[3]	JAVA based Web Application	SQL Injection
VIPER[4]	ANY	SQL Injection
Our Proposed IDS	ANY	SQL Injection, XSS attacks, Command Injection, Directory Traversal Attacks

It is clear from the comparison that the proposed IDS will support to all web applications which are developed using any language like java, PHP, Dot Net etc. The proposed IDS will also detect to the SQL Injection attacks, XSS attacks, Directory Traversal attacks and Command injection attacks. Previous IDS system ARDILLA [2] do not detect command injection attacks, directory traversal attacks and they perform detection on only on PHP based web application. WASP [3] do not detect XSS attacks; command injection attacks and directory traversal attacks and they perform detection only on java based web application. And another tool is VIPER [4] do not detect XSS attacks, command injection attacks and directory traversal attacks.

6. Conclusion

In this paper, an enhanced intrusion detection system for input validation attacks on web application is proposed. The proposed intrusion detection system detect SQL injection attacks, XSS attacks, command injection attacks and also detect to directory traversal attacks. In addition with this the proposed IDS support all web applications which are developed using any language like java, php, Dot Net etc. Web application security can be improved with proposed IDS by making security provision active in advance .The analysis time to detect IVA is reduced in this IDS because whole process is operated without developer intervention.

References

[1] OWASPD-Open Web Application Security Project. “Top ten most critical Web Application Security Risks”, https://www.owasp.org/index.php/Top_10_2010-Main.

[2] Adam Kie˙zun,Philip J. Guo,Karthick Jayaraman,Michael D. Ernst:“Automatic Creation of SQL Injection and Cross-Site Scripting AttackS”, ICSE‟09, May 16-24, 2009, Vancouver, Canada.

[3] William G.J. Halfond, Alessandro Orso.” WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation” IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 34, NO. 1, JANUARY/FEBRUARY 2008.

[4] Angelo Ciampa, Corrado Aaron Visaggio, Massimiliano Di Penta :”A heuristic-based approach for detecting SQL-injection vulnerabilities in Web applications”. *SESS’10* , May 2, 2010, Cape Town, South Africa Copyright 2010 ACM.

[5] William G.J. Halfond and Alessandro Orso,” Preventing SQL Injection Attacks Using AMNESIA” ICSE’06, May 20–28, 2006, Shanghai, China ACM 06/0005.

[6] Takeshi Matsuda,Daiki Koizumi,Michio Sonoda,Shigeichi Hirasawa, ”On predictive errors of SQL injection attack detection by the feature of the single character” Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on 9-12 Oct 2011, On Page 1722-1727.

[7] Raju haldar and Agostino Cortesi, “Obfuscation-based Analysis of SQL Injection Attacks” IEEE.

[8] C# Network Programming by Richard Blum ISBN: 0782141765, Sybex@2003.
[9] Y.Xie and A. Aiken." Static detection of Security vulnerabilities in scripting language", In USENIX.2006.
[10] A. Sabelfeld and A. Myers."Language-based information flow security". Selected Areas in Communications 2003.
[11] R.McClure and I. Kruger."SQL DOM: compile time checking of dynamic SQL statements". In ICSE, 2005.
[12] M. Martin and M. Lam."Automatic generation of XSS and SQL injection attacks with goal-directed model checking". In USENIX SEcurity,2008.
[13] S. Bandhakavi. P. Bisht, P. Madhusudan, and V.N. Venkatakrishan."CANDID: preventing SQL injection attacks using dynamic candidate evaluations". In CCS,2007.
[14] Z. Su and G. Wassermann,"The Essence of Command Injection Attacks in Web Applications". Proc.33rd Ann. Symp. Principles of Programming Language, p.372-382, Jan 2006.
[15] D. Scott and R. Sharp,"Avstracting Application-Level Web Security", Proc. 11th Int'l Conf. World Wide Web, p.396-407, May 2002.
[16] W. R. Cook and S. Rai, "Safe Query Objects: Statically Typed Objects as Remotely Executable Queries" Proc. 27th Int'l Conf. Software Eng. p.97-106, May2005.

First Author Puspendra Kumar perusing Master of technology in Information security from Maulana Azad national Institute of Technology Bhopal.

Second Author Dr. R.K. Pateriya working as Associate Professor in computer science and engineering department of Maulana Azad National Institute of Technology, Bhopal.