



SQL Injection & XSS Vulnerability Detection and Prevention in Web Application

Priyank Bhojak¹, Nikita Patel², Chintan Patel³, Jatin Patel⁴

¹IT Department, BVM priyankbhojak@gmail.com

²IT Department, BVM nikita.patel@bvmengineering.ac.in

³CE Department, Marwadi Education chintan.p592@gmail.com

⁴CE Department name, GPERI jatinit2010@gmail.com

Abstract — *As the popularity of the web increases and web applications become tools of everyday use, the role of web security has been importance as well. Last few years have shown a significant increase in the number of web-based attacks. Web sites are dynamic, static, and most of the time a combination of both. Web sites need protection in their database to assure security. Web scanner is a tool designed to discover security holes in your web applications that an attacker can access to your systems and data. It looks for multiple vulnerabilities including SQL injection, cross site scripting and weak passwords etc. This paper demonstrates how easy it is for attackers to automatically discover and exploit web application-level vulnerabilities in a large number of web applications.*

Keywords:- *Web Scanner, SQL Injection, Cross Site Scripting, web crawler, Input Vector, Web application vulnerability*

1. INTRODUCTION

In computer security, the term vulnerability is applied to a weakness in a system that allows an attacker to violate the integrity of that system [1]. Vulnerabilities may result from weak passwords, software bugs, a computer virus or a script code injection, and a SQL injection.

Web sites are dynamic, static, and most of the time a combination of both. Websites need protection in their database to assure security. An SQL injection attacks interactive web applications that provide database services. These applications take user inputs and use them to create an SQL query at run time. In an SQL injection attack, an attacker might insert a malicious SQL query as input to perform an unauthorized database operation. Using SQL injection attacks, an attacker can retrieve or modify confidential and sensitive information from the database. It may jeopardize the confidentiality and security of Web sites which totally depends on databases. This report presents a “code reengineering” that implicitly protects the applications which are written in PHP from SQL injection attacks. It uses an original approach that combines static as well as dynamic analysis. [2].

A common break-in strategy is to try to access sensitive information from a database by first generating a query that will cause the database parser to malfunction, followed by applying this query to the desired database. Such an approach to gaining access to private information is called SQL injection. Since databases are everywhere and are accessible from the internet, dealing with SQL injection has become more important than ever.

Detecting vulnerabilities is generally not an easy task, and not all of the common vulnerabilities can be successfully detected by automated scanners [2]. There are two main approaches to testing software applications for the presence of bugs and vulnerabilities [8].

In white-box testing, the source code of the application is analyzed in an attempt to track down defective or vulnerable lines of code. This operation is often integrated into the development process by creating add-on tools for common development environments. So far, white-box testing has not experienced widespread use for finding security flaws in web applications.[8] An important reason is the limited detection capability of white-box analysis tools, in particular due to heterogeneous programming environments and the complexity of applications that incorporate database, business logic, and user interface components.

In black-box testing, the source code is not examined directly. Black box testing is the Software testing method which is used to test the software without knowing the internal structure of code or program. Most likely this testing method is what most of tester actual perform and used the majority in the practical life.

The main contributions of this paper are that we demonstrate how easy it is for attackers to automatically discover and exploit application-level vulnerabilities in a large number of web applications. Detecting vulnerabilities is generally not

an easy task, and not all of the common vulnerabilities can be successfully detected by automated scanners. In addition, this paper helps you to suggest areas for Web Vulnerability Scanner tool improvement.

2. SQL INJECTION ATTACK

SQL injection attacks are based on injecting strings into database queries that alter their intended use. This can occur if a web application does not properly filter user input [8]. There are many varieties of SQL. Most dialects are loosely based on the most recent ANSI standard SQL-92. The typical unit of execution in the SQL language is the query, a collection of statements that are aimed at retrieving data from or manipulating records in the database. A query typically results in a single result set that contains the query results. Apart from data retrieval and updates, SQL statements can also modify the structure of databases using Data Definition Language statements (“DDL”).

A web application is vulnerable to an SQL injection attack if an attacker is able to insert SQL statements into an existing SQL query of the application. This is usually achieved by injecting malicious input into user fields that are used to compose the query. For example [7], consider a web application that uses a query such as the one shown in following for authenticating its users.

```
SELECT ID,LastLogin FROM Users WHERE User = 'efg' AND password = 'xyz '
```

This query retrieves the ID and LastLogin fields of user “efg” with password “xyz” from table Users. Such queries are typically used for checking the user login credentials and, therefore, are prime targets for an attacker. In this example, a login page prompts the user to enter her username and password into a form. When the form is submitted, its fields are used to construct an SQL query that authenticates the user.

```
sqlQuery = "SELECT ID , LastLogin FROM Users WHERE User = " + userName + " AND Password = " + password + """
```

If the login application does not perform correct input validation of the form fields, the attacker can inject strings into the query that alter its semantics. For example, consider an attacker entering user credentials such as

```
User: ' OR 1=1 --  
Password :
```

Using the provided form data, the vulnerable web application constructs a dynamic SQL query for authenticating the user as shown in

```
SELECT ID, LastLogin FROM Users WHERE User = 'OR 1=1 -- AND Password = '
```

The “--” command indicates a comment in Transact- SQL. Hence, everything after the first “--” is ignored by the SQL database engine. With the help of the first quote in the input string, the user name string is closed, while the “OR 1=1” adds a clause to the query which evaluates to true for every row in the table. When executing this query, the database returns all user rows, which applications often interpret as a valid login.

2.1. SQL Injection Discovery Technique:

It is not compulsory for an attacker to visit the web pages using a browser to find if SQL injection is possible on the site. Generally attackers build a web crawler to collect all URLs available on each and every web page of the site. Web crawler is also used to insert illegal characters into the query string of a URL and check for any error result sent by the server. If the server sends any error message as a result, it is a strong positive indication that the illegal special meta character will pass as a part of the SQL query, and hence the site is open to SQL Injection attack. For example Microsoft Internet Information Server by default shows an ODBC error message if an any meta character or an unescaped single quote is passed to SQL Server. The Web crawler only searches the response text for the ODBC messages.

A. Preventing SQL injection method

Andrey Petukhov & Dmitry Kozlov. Explained in detail about the methods which are used to prevent an SQL injection attacks. [2]

- 1) Static analysis
- 2) Run time analysis

These techniques are based on the stored procedures, Authors' has used control flow graph that notifies what user inputs to the dynamic built SQL statement. Control flow graphs are very useful to minimize the set of SQL statements to verify users input. In run time analysis we access information about stored statement from Finite State Automaton to narrow the verification procedure and to indicate the user's inputs true or false. [2]

3. CROSS-SITE SCRIPTING (XSS) ATTACK

Cross-Site Scripting or XSS allows attackers to inject client-side script in a web page [7]. The attacker injects script, such as JavaScript, VBScript, ActiveX, HTML, or flash into an application to try to get access to sensitive information Dynamic websites (using AJAX, Flex, for example) are vulnerable. Static websites are not at risk [8].

XSS attacks are generally simple to execute, but difficult to prevent and can cause significant damage. There exist two different types of XSS attacks: *reflected* and *stored* XSS attacks [7].

The most common one found in web applications today is called reflected XSS attack. Unfortunately, the search form on the web site fails to perform input validation, and whenever a search query is entered that does not return any results, the user is displayed a message that also contains the unfiltered search string. For example [8], if the user enters a search string "<i>Hello World</i>", the italics markers (i.e., <i>) are not filtered, and the browser of the user displays "No matches for Hello World" (note that the search string is displayed in italics).

This indicates that there is a reflected XSS vulnerability present in the application, which can be exploited in the following way. First, an attacker writes a JavaScript snippet that, when executed in a victim's browser, sends the victim's cookie to the attacker. Now, the attacker tricks the victim into clicking a link that points to the action target of the vulnerable form and contains the malicious script as URL (GET) parameter.

www.india-banking .com/search .php?searchterm ={ attacker's script goes here}

When the user clicks on this link, the vulnerable application receives a search request similar to the previous one, where the search term was <i>Hello World</i>. The only difference is that now, the search term is the malicious script written by the attacker. Instead of a harmless phrase in italics, the victim's browser now receives malicious JavaScript code from a trusted web server and executes it. As a result, the user's cookie, which can contain authentication credentials, is sent to the attacker.

This example also makes clear why the attack is called reflected; the malicious code arrives at the victim's browser after being reflected back by the server.

The second type of XSS attack is called stored XSS attack [8]. As its name suggests, the difference compared to the reflected attack is that the malicious script is not immediately reflected back to the victim by the server, but stored inside the vulnerable application for later retrieval.

Here in following is example of an XSS attack string which generates a page with arbitrary content on an XSS vulnerable site:

http://www.VulnerableSite.com/search?q=<iframestyle=height:100%;width=100%;border:none;transparent:none;position:absolute;top:0;left:0;src=http://www.AttackerSite.com/ >

The attack string can be URL encoded so that the content is unreadable for the average Internet user. An attack is successful if a victim visits an URL containing the XSS attack. This can be achieved by e.g. E-mail from a sender in which the user has trust.

4. DETECTION OF SQL INJECTION & XSS VULNERABILITY

Vulnerability scanner consists of four components. First the Crawling component that gathers a set of target web sites. Second Attack component launches the configured attacks against these targets. Third Analysis component examines the results returned by the web applications to determine whether an attack was successful. Finally Generate Report.

The web crawler interacts with web applications, and gathers information (e.g. web pages, form information) for detection engine. Detection engine constructs web request with some specified attacking code. Detection engine waits for the response and analyzes it, once the specified key words can be detected in the responded data, the vulnerability is identified.

4.1 Detecting Vulnerabilities

The most efficient way of finding vulnerabilities in web applications is manual code review. This technique is very time-consuming, requires expert skills, and is prone to overlooked errors. Therefore, security society actively develops automated approaches to finding security vulnerabilities. These approaches can be divided into two wide categories: black-box testing and white-box testing.

White-box analysis consists of examining the code without executing it. Developers can do this in one of two ways [4]: manually, during code inspections and reviews; or automatically, using automated analysis tools. Peers systematically examine the delivered code, searching for programming mistakes. Security inspections are the most effective way to minimize vulnerabilities in an application; they are a crucial procedure when developing software for critical systems.

Black-box testing [7] refers to the analysis of program execution from an external point of view. In short, it consists of comparing the software execution outcome with the expected result. Testing is probably the most used technique for software verification and validation.

4.2 Attack Component

The attack component scans each page for the presence of web forms. The reason is that the fields of web forms constitute our entry points to web applications.

For each web form, we extract the action and the method (i.e., GETS or POST) used to submit the form content. Also, the form fields and its corresponding parameters are collected. Then, depending on the actual attack that is launched, appropriate values for the form fields are chosen. Finally, the form content is uploaded to the server specified by the action address (using either a GET or POST request). After that attacked server responds to such a web request by sending back a response page via HTTP.

4.3 Penetration Testing

Penetration testing approach is based on simulation of attacks against web applications. Currently, penetration testing is implemented as black box testing. Thus the Scope of analysis is limited to HTTP responses. Actually black box penetration testing is working as following process [2]:

1. The first step is to identify all pages being part of the web application. This phase is crucial for black box testing as attacks could be launched only against recognized application Data Entry Points. This task can be fulfilled automatically (Using web crawlers),
2. The second step is to extract Data Entry Points (DEPs) from pages visited in the first step. The result is a set of DEPs to be analyzed.
3. The third step is simulation of attacks. Every parameter in every DEP is fuzzed with malicious patterns and used within an HTTP request to web application.
4. Finally every received HTTP response is scanned for indications of vulnerability. [2]

4.4 Attack Component

The attack component first constructs a web request and sends it to the target application, using a simple script as input to each form field. The server processes the request and returns a response page. This response page is analyzed for occurrences of the injected script code. For detecting vulnerability, this simple variant of a XSS attack uses plain JavaScript code as shown in following.

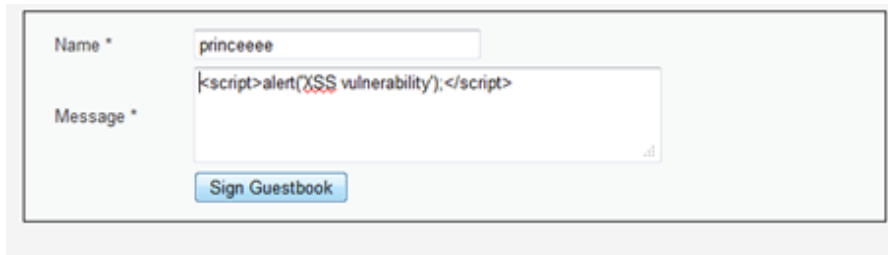
```
<script>alert('XSS vulnerability ');</script>
```

If the target web form performs some kind of input validation and filters quotes or brackets, this attack will fail. The simple XSS analysis module takes into account that some of the required characters for scripting (such as quotes or brackets) could be filtered or escaped by the target web application.

The first response from following insecure code (low security level source code) shows an example of the result page that contains XSS vulnerability in the response.

```
<?php
    if(isset($_POST['btnSign']))
    {
        $message = trim($_POST['mtxMessage']);
        $name = trim($_POST['txtName']);
        // Sanitize message and name input
        $message = mysql_real_escape_string($message);
        $name = mysql_real_escape_string($name);

        $query = "INSERT INTO guestbook (comment,name) VALUES
        ('$message','$name)";
        $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre>');
    }
?>
```



Listing 1. Low security level source code that contain xss vulnerability

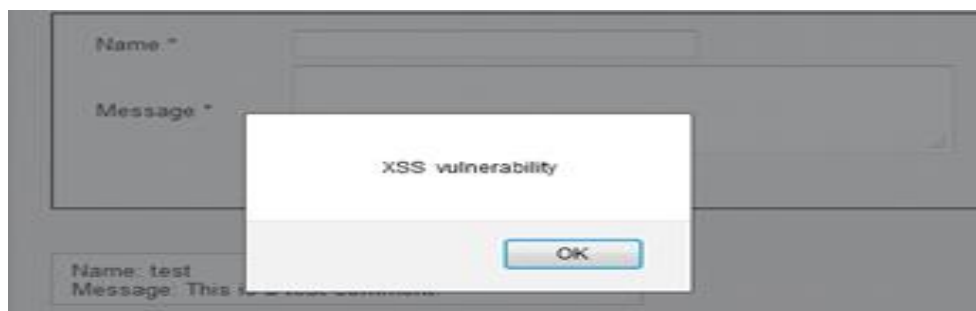


Figure 1: Injected XSS vulnerability into page and show alert on response

As you can see from the above Figure 1, we have successfully injected a XSS payload into the database. In this example we used the '<script>alert('XSS vulnerability ');</script>' payload within the \$message variable. If we take a look at the high security level source code for the same vulnerability it should give us some clues as to why the low security level is insecure.

The second response page is an example of highly secure code (shown in Listing 2.), here we have two variables passed from the form which contains user supplied input, these are \$name and \$message. The first thing we do is use the trim() PHP function to remove any white space from the beginning or end of the strings. The \$message variable is passed through the stripslashes() PHP function to remove any slashes and then also passed through the mysql_real_escape_string() PHP function to escape any special characters, this prevents from SQL Injection and XSS. The \$name variable is only passed through the mysql_real_escape_string() function before being placed in the final query (\$query). So as you can see there has been some input sanitization. Thus, the script will not be executed as it is not correctly embedded within the HTML page.

```
<?php
if(isset($_POST['btnSign']))
{
    $message = trim($_POST['mtxMessage']);
    $name = trim($_POST['txtName']);
    // Sanitize message input
    $message = stripslashes($message);
    $message = mysql_real_escape_string($message);
    $message = htmlspecialchars($message);
    // Sanitize name input
    $name = stripslashes($name);
    $name = mysql_real_escape_string($name);
    $name = htmlspecialchars($name);

    $query = "INSERT INTO guestbook (comment,name) VALUES
('$message','$name)";
    $result = mysql_query($query) or die('<pre>' . mysql_error() . '</pre>');
```

Listing 2. Highly secure code which not contain xss vulnerability

If you compare the low security level source code to the high security level one you will notice that the high security level source code has some extra input sanitisation. Both the \$name and \$message variables are passed through the htmlspecialchars() PHP function. The htmlspecialchars() function converts special characters to HTML entities, therefore the user input is HTML encoded meaning that it is just displayed as normal HTML. The following characters are affected:

- & (ampersand) becomes &
- " (double quote) becomes "
- ' (single quote) becomes '
- < (less than) becomes <
- > (greater than) becomes >

4.5 Authorization Bypass using SQL command Injection

Auth Bypass flaw comes up every time a website doesn't filter the attackers input. It deals with Sql command injection. For example the target website uses this vulnerable, unsecured authorization script:

```
<?php
$sql = "SELECT * FROM users WHERE username='" . $_POST['username'] . "'
AND password='" . $_POST['password'] . "'";
response = mysql_query($sql);
?>
```


As you can see, the user's input is not getting checked or filtered. This is how the MySQL Query looks now:

```
SELECT * FROM users WHERE user="" AND password=""
```

Let's take a simple username (mostly admin or administrator) and as a password, we choose:

```
' OR 'a' = 'a
```

Now this is how the MySQL Query looks now:

```
SELECT * FROM users WHERE user='admin' AND password="" OR 'a' = 'a'
```

'a' = 'a' is a true value, just like 1 = 1 or 'cats' = 'cats'. Let's analyse the situation in words:

```
Username='admin' AND Password="" OR 'a' = 'a'  
means -> Username admin and Password TRUE
```

So now the MySQL Query looks now:

```
SELECT * FROM users WHERE user='admin' AND TRUE
```

That means we're getting logged in as the administrator, without a password by manipulating the query.



Figure 2: Insert data for Authorization Bypass using SQL command Injection

One of the methods to fix and secure such Auth Bypass flaws, is to use the php function `mysql_real_escape_string`. It causes that every of these characters:

`\x00, \n, \r, \, '` get's replaced with a simple Backslash `\"`, so the attacker's commands become useless.

```
// Secure Code  
<?php  
$username = mysql_real_escape_string($_POST["username"]);  
$password = mysql_real_escape_string($_POST["password"]);  
$sql = "SELECT * FROM users WHERE username=\"" . $username . "\" AND password=\"" .  
$password . "\"";  
$response = mysql_query($sql);
```

4.6 Analysis component and Generate Report

After an attack has been launched, the analysis component has to parse and interpret the server response. An analysis component uses attack-specific response criteria and keywords to calculate a confidence value to decide if the attack was successful or any false positives are possible. At last generate report of which vulnerability found by penetrating testing in the web page. And report will generate category-wise and vulnerability-wise and risk level-wise.

5. CONCLUSIONS AND FUTURE WORKS

The main contribution of this research paper is to show how easy it is to automatically discover and exploit web application- level vulnerabilities in a large number of web applications. Many web application security vulnerabilities result from generic input validation problems. Examples of such vulnerabilities are SQL Injection and Cross-Site Scripting (XSS). Although the majority of web vulnerabilities are easy to understand and avoid, many web developers are unfortunately not security-aware and there is general consensus that there exist a large number of vulnerable applications and web sites on the web [10]. Research proposed the flow of web vulnerability scanner that analyzes web sites for exploitable SQL and XSS vulnerabilities. Automated Vulnerability Detection method based on web crawling is proposed in this research paper. To the end, this paper helps you to suggest areas for Web Vulnerability Scanner tool improvement and it allows developer to develop an extensive good scanner.

In the future, our research will includes improving on detecting web security vulnerabilities. In order to build a better SQL injection and XSS vulnerability detection approach and more works on studying the complex-form analyzing, the attacking codes constructing and the response analyzing.

6. ACKNOWLEDGMENT

We are thankful to Department Of Information Technology, Birla Vishvakarma Mahavidyalaya , Vallabh Vidhyanagar, India for their support and for providing necessary guidance concerning projects implementation. We are also thankful to Mr.kaushal bhavsar , Pratikar technology for providing guidance of security tools and implementation.

7. REFERENCES

- [01] V. Suhina, S. Groš, Z. Kalafatić, Detecting vulnerabilities in Web applications by clustering Web pages (pp. 01-07) , Faculty of Electrical Engineering and Computing, University of Zagreb , Croatia
- [02] Andrey Petukhov, Dmitry Kozlov. (2008) . Detecting Security Vulnerabilities in Web Applications Using Dynamic Analysis with Penetration Testing (pp. 01-05) , Dept. of Computer Science, Moscow State University.
- [03] Xin Wang, Luhua Wang, Gengyu Wei, Dongmei Zhang, Yixian Yang. (2010). Hidden Web Crawling For Sql Injection Detection (pp. 14-18) , Published by the IEEE. (978-1-4244-6769-3/10)
- [04] Nuno Antunes , Marco Vieira. (2012). Defending against Web Application Vulnerabilities (pp. 66-72) , Published by the IEEE Computer Society. 0018-9162/12. Volume.-2.
- [05] Jeremiah Grossman WhiteHat Security founder & CTO Website Vulnerabilities Revealed (pp. 08-14). WhiteHat Security. (2008)
- [06] Vebjørn Moen, Andr´e N. Klingsheim, Kent Inge Fagerland Simonsen, Kjell Jørgen Hole. Vulnerabilities In E-governments. (pp. 01-04). University of Bergen.
- [07] Dafydd Stuttard, Marcus Pinto. (2011). The Web application Hacker’s Handbook Finding an Exploiting Security Flaws. second edition.
- [08] Stefan Kals, Engin Kirda, Christopher Kruegel, Nenad Jovanovic. SecuBat: A Web Vulnerability Scanner. Secure Systems Lab, Technical University of Vienna.
- [09] David Shelly, Randy Marchany, Joseph Tront. (2010). Closing the Gap: Analyzing the Limitations of Web Application Vulnerability Scanners. Virginia Polytechnic Institute and State University
- [10] Katkar Anjali S , Kulkarni Raj B. (2012) Web Vulnerability Detection and Security Mechanism . (pp. 237-241). International Journal of Soft Computing and Engineering (IJSCE). ISSN: 2231-2307, Volume-2.
- [11] Kanganand Monika. Web Application Vulnerabilities and Detection. (pp. 02-05). U.I.E.T, Punjab University, Chandigarh, U.T, India
- [12] Marco Vieira, Nuno Antunes, Henrique Madeira CISUC. Using Web Security Scanners to Detect Vulnerabilities in Web Services Department of Informatics Engineering University of Coimbra – Portugal
- [13] Acunetix WVS (2004) . Acunetix web vulnerability scanner a real world review (pp. 02-20) Available at <http://www.acunetix.com>
- [14] William G.J. Halfond, Shauvik Roy Choudhary, Alessandro Orso..Penetration Testing with Improved Input Vector Identification. (pp. 01-03). College of Computing ,Georgia Institute of Technology.