A Research on Efficient Secure Coding Method for Preventing Web Applications from Different Types of Attacks

Kunal D. Garud [1], Krunal J. Panchal [2]

[1] *Student, Computer Engineering,* LJ Institute of Engineering & Technology, *Gujarat, India*
[2] *Prof., Computer Engineering,* LJ Institute of Engineering & Technology, *Gujarat, India*

ABSTRACT

Web applications provide vast category of functionalities and usefulness. As more and more sensitive data is available over the internet hackers are becoming more interested in such data revealing which can cause massive damage. SQL injection is one of such attacks. This attack can be used to infiltrate the database of any web application that may lead to alteration of database or disclosing important information. Cross site scripting is one more attack in which attacker obfuscates the input given to the web application that may lead to changes in view of the web page. So, proposed model in this literature can be very useful to prevent web application from this type attacks. where opinions are highly unstructured.

Keyword - *Web Application Security, Cross Site Scripting (XSS), SQL Injection, Path Traversal, Remote File Inclusion, OS command Injection, Web Application Vulnerability, OWASP Top 10, Secure Coding*

1. INTRODUCTION

1) Cross-Site Scripting (also known as XSS) is one of the most common application-layer web attacks. XSS vulnerabilities target scripts embedded in a page that are executed on the client-side (in the user's web browser) rather than on the server-side. XSS in itself is a threat that is brought about by the internet security weaknesses of client-side scripting languages, such as HTML and JavaScript. The concept of XSS is to manipulate client-side scripts of a web application to execute in the manner desired by the malicious user. Such a manipulation can embed a script in a page that can be executed every time the page is loaded, or whenever an associated event is performed.
XSS is the most common security vulnerability in software today. This should not be the case as XSS is easy to find and easy to fix. XSS vulnerabilities can have consequences such as tampering and sensitive data theft.

Key Concepts of XSS

1. XSS is a web-based attack performed on vulnerable web applications.
2. In XSS attacks, the victim is the user and not the application.
3. In XSS attacks, malicious content is delivered to users using JavaScript.

Explaining Cross-Site Scripting

An XSS vulnerability arises when web applications take data from users and dynamically include it in web pages without first properly validating the data. XSS vulnerabilities allow an attacker to execute arbitrary commands and display arbitrary content in a victim user's browser. A successful XSS attack leads to an attacker controlling the victim's browser or account on the vulnerable web application. Although XSS is enabled by vulnerable pages in a web application, the victims of an XSS attack are the application's users, not the application itself. The potency of an

XSS vulnerability lies in the fact that the malicious code executes in the context of the victim's session, allowing the attacker to bypass normal security restrictions.

Reflective XSS

There are many ways in which an attacker can entice a victim into initiating a reflective XSS request. For example, the attacker could send the victim a misleading email with a link containing malicious JavaScript. If the victim clicks on the link, the HTTP request is initiated from the victim's browser and sent to the vulnerable web application. The malicious JavaScript is then reflected back to the victim's browser, where it is executed in the context of the victim user's session.

Persistent XSS

Consider a web application that allows users to enter a username that is displayed on each user's profile page. The application stores each username in a local database. A malicious user notices that the web application fails to sanitize the username field and inputs malicious JavaScript code as part of their username. When other users view the attacker's profile page, the malicious code automatically executes in the context of their session.

2) SQL injection (SQLi)

SQL injection (SQLi) is an application security weakness that allows attackers to control an application's database – letting them access or delete data, change an application's data-driven behaviour, and do other undesirable things – by tricking the application into sending unexpected SQL commands.

SQL injection weaknesses occur when an application uses untrusted data, such as data entered into web form fields, as part of a database query. When an application fails to properly sanitize this untrusted data before adding it to a SQL query, an attacker can include their own SQL commands which the database will execute. Such SQLi vulnerabilities are easy to prevent, yet SQLi remains a leading web application risk, and many organizations remain vulnerable to potentially damaging data breaches resulting from SQL injection.

How Attackers Exploit SQLi Vulnerabilities:

Attackers provide specially-crafted input to trick an application into modifying the SQL queries that the application asks the database to execute. This allows the attacker to:

1. Control application behaviour that's based on data in the database, for example by tricking an application into allowing a login without a valid password
2. Alter data in the database without authorization, for example by creating fraudulent records, adding users or "promoting" users to higher access levels, or deleting data
3. Access data without authorization, for example by tricking the database into providing too many results for a query

Anatomy of a SQL Injection Attack

A developer defines a SQL query to perform some database action necessary for their application to function. This query has an argument so that only desired records are returned, and the value for that argument can be provided by a user (for example, through a form field, URL parameter, web cookie, etc.).

A SQLi attack plays out in two stages:

1. Research: Attacker tries submitting various unexpected values for the argument, observes how the application responds, and determines an attack to attempt.
2. Attack: Attacker provides a carefully-crafted input value that, when used as an argument to a SQL query, will be interpreted as part of a SQL command rather than merely data; the database then executes the SQL command as modified by the attacker.

3) Remote File Inclusion:

Remote File Include (RFI) is an attack technique used to exploit "dynamic file include" mechanisms in web applications. When web applications take user input (URL, parameter value, etc.) and pass them into file include commands, the web application might be tricked into including remote files with malicious code.

An attacker can use RFI for:-
Running malicious code on the server: Any code in the included malicious files will be run by the server. If the file include is not executed using some wrapper, code in include files is executed in the context of the server user. This could lead to a complete system compromise.
Running malicious code on clients: The attacker's malicious code can manipulate the content of the response sent to the client. The attacker can embed malicious code in the response that will be run by the client (for example, JavaScript to steal the client session cookies).

4) Directory Traversal:-
A directory traversal (or path traversal) consists in exploiting insufficient security validation / sanitization of user-supplied input file names, so that characters representing "traverse to parent directory" are passed through to the file APIs.
The goal of this attack is to use an affected application to gain unauthorized access to the file system. This attack exploits a lack of security (the software is acting exactly as it is supposed to) as opposed to exploiting a bug in the code
In web applications with dynamic pages, input is usually received from browsers through GET or POST request methods. Here is an example of an HTTP GET request URL
GET http://test.webarticles.com/show.asp?view=oldarchive.html HTTP/1.1
Host: test.webarticles.com
The attacker would assume that show.asp can retrieve files from the file system and sends the following custom URL.
GET http://test.webarticles.com/show.asp?view=../../../../../Windows/system.ini HTTP/1.1
Host: test.webarticles.com
This will cause the dynamic page to retrieve the file system.ini from the file system and display it to the user. The expression.../ instructs the system to go one directory up which is commonly used as an operating system directive. The attacker has to guess how many directories he has to go up to find the Windows folder on the system, but this is easily done by trial and error

5) Command Injection:-
An Operating System (OS) command injection attack occurs when an attacker attempts to execute system level commands through a vulnerable web application. Applications are considered vulnerable to the OS command injections if they can be manipulated into executing unauthorized system commands via the web interface
If the weakness occurs in a high privileged program, it may allow an attacker to specify commands that would not be available otherwise, or call alternate commands with privileges that the attacker does not have.The danger of this weakness is aggravated if the breached process does not follow the principle of the least privilege because the malicious command may gain system privileges and allow the attacker to cause additional damage.

2. RELATED WORK

[1] Predicting Cross-Site Scripting (XSS) Security Vulnerabilities in Web Applications

In this paper proposed system is based on machine learning algorithms so it is efficient but it is difficult to implement and execute this type of algorithms.

[2] Removing Cross-Site Scripting Vulnerabilities from Web Applications using the OWASP ESAPI Security Guidelines

In this paper XSS prevention rules according to OWASP are suggested. This rules are very useful for preventing web applications. But it is required to implement methodology which can utilize this rules for preventing web applications from XSS.

[3] Research of SQL Injection Attack and Prevention Technology.

In this paper they have proposed SQL Defense model using which we can prevent web application from SQL Injection.

The SQL Defense model is efficient for discarding malicious input. But it can not be applied to web applications dynamically in live environment.

[4] SQL Injection Attack Prevention Based on Decision Tree Classification

In this paper they have proposed a SQL engine which is defending SQL Injection using Decision tree. But this model increases processing overhead of web application for request-response architecture.

[5] A Sound Framework for Dynamic Prevention of Local File Inclusion

This framework ensures Directory Traversal resistance when applied before the web deployment in a web server i.e. when there is no additional script in web application folders and subfolders.

[6] Commix: Detecting and Blocking command injection flaws

In this paper they have proposed a commix: Software Architecture which will generate attack vectors of OS Command injection and this attacks will be detected by Vulnerability detection module and blocks the attack to execute any command on server

[7] A Novel Approach for Detection of SQL Injection and Cross Site Scripting Attacks

Whenever user will make a web request instead of directly generating a SQL query for directly we are processing this request for any kind of SQL injection attacks or XSS attacks based on input parameters. This model implements a strong input distillation. Input checking is a very critical in case of XSS attacks. Before passing the XSS infected web request directly to the database where it can get permanently stored on data logic of web application, this model tries to inform the admin of possible XSS or SQL injection attacks.

3. COMPARATIVE ANALYSIS

Sr. No.	Paper Title	Published Year	Method	Disadvantage
1	Predicting Cross-Site Scripting (XSS) Security Vulnerabilities in Web Applications	IEEE 2015	Machine learning classifier	Difficult to implement algorithms
2	Removing Cross-Site Scripting Vulnerabilities from Web Applications using the OWASP ESAPI Security Guidelines	INDJST 2015	Rules based approach	Could not applied to web application dynamically
3	Research of SQL Injection Attack and Prevention Technology	ICEDIF 2015	SQL defense model	Low performance
4	SQL Injection Attack Prevention Based on Decision Tree Classification	IEEE 2014	Three tier architecture using Decision trees	Difficult to implement query engine

5	A Novel Approach for Detection of SQL Injection and Cross Site Scripting Attacks	IEEE 2015	Dynamic input processing	Need to improve for detection and prevention.
6	A Sound Framework for Dynamic Prevention ofLocal File Inclusion	IEEE 2015	AntiLFIer	Language Specific
7	Commix: Detecting and Blocking command injection flaws	UPRC 2015	Commix Architecture	Difficult to use.

4. PROBLEM STATEMENT

The fundamental security problem with web applications is that all user inputs are untrusted this gives rise to number of security breaches. A huge variety of attacks against web applications involve submitting unexpected input, crafted to cause behaviour that was not intended by the application's designers. Input-based vulnerabilities can arise anywhere within an application's functionality,
and in relation to practically every type of technology in common use. So, It is required to prevent web applications from this type of malicious inputs.

5. PROPOSED MODEL

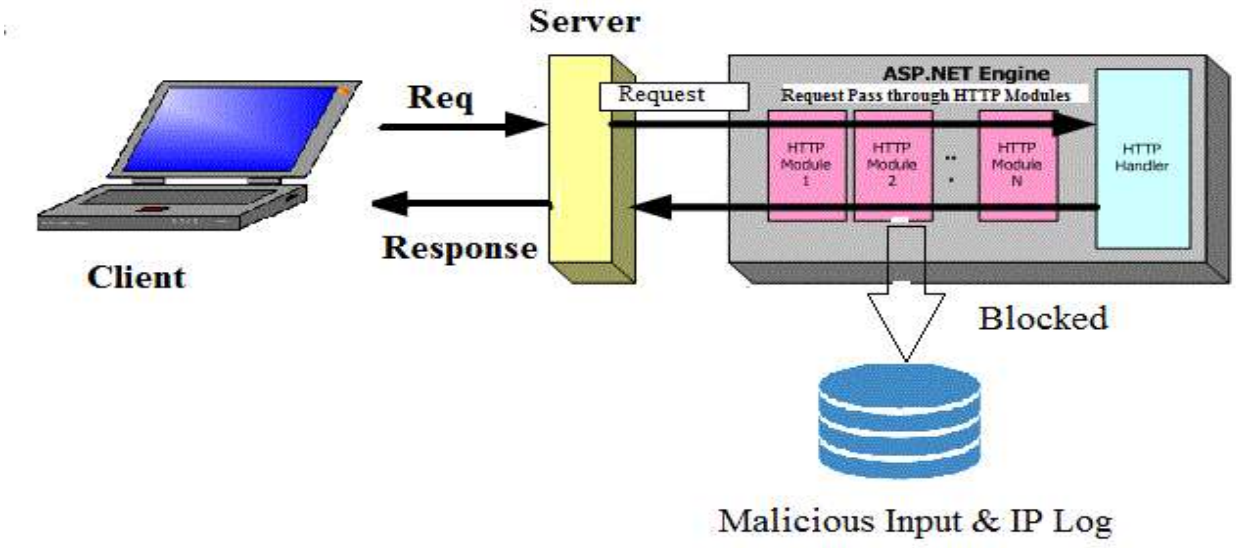

Figure 1. Proposed Flow Model.

6. ANALYSIS

There are four possible test outcomes in the Benchmark:

- Correctly identifies a real vulnerability (True Positive - TP)
- Fails to identify a real vulnerability (False Negative - FN)
- Correctly ignores a false alarm (True Negative - TN)
- Fails to ignore a false alarm (False Positive - FP)

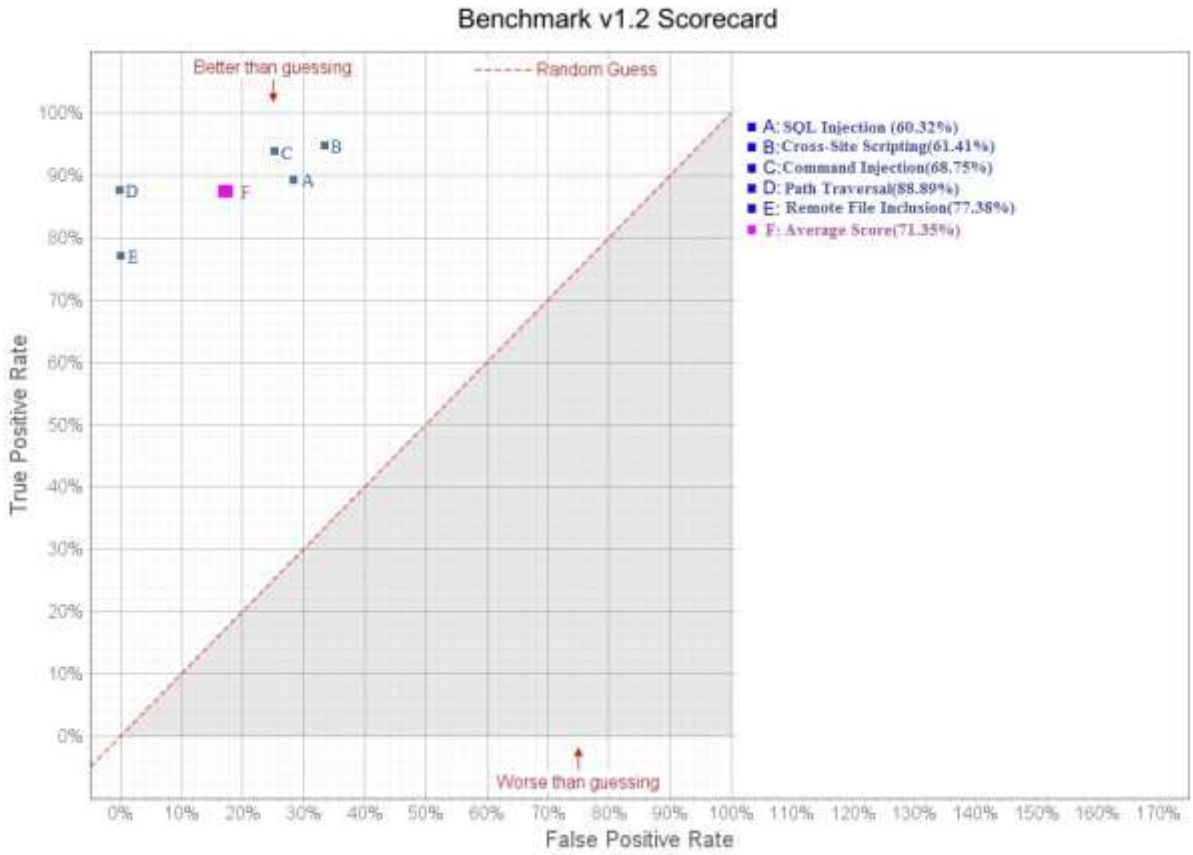

Fig 2 Accuracy Scorecard

Key:

True Positive (TP)	Tests with real vulnerabilities that were correctly reported as vulnerable.
False Negative (FN)	Tests with real vulnerabilities that were not correctly reported as vulnerable.
True Negative (TN)	Tests with fake vulnerabilities that were correctly not reported as vulnerable.
False Positive (FP)	Tests with fake vulnerabilities that were incorrectly reported as vulnerable.
True Positive Rate (TPR) = TP / (TP + FN)	The rate at which the correctly reports real vulnerabilities.
False Positive Rate (FPR) = FP / (FP + TN)	The rate at which the system incorrectly reports fake vulnerabilities as real.
Score = TPR - FPR	Normalized distance from the random guess line.

7. CONCLUSIONS

Web application Security starts with the Architecture and Design. Security can't be added on later without re-designing and rewriting. Custom code often introduces vulnerabilities. Web application vulnerabilities are not prevented by traditional security controls. So, making a secure code solution which can be applicable to every web application will be very essential.

8. REFERENCES

[1] Mukesh Kumar Gupta, Mahesh Chandra Govil, Girdhari Singh (2015), "**Predicting Cross-Site Scripting (XSS) Security Vulnerabilities in Web Applications**" , 2015 IEEE, 12th International Joint Conference on Computer Science and Software Engineering (JCSSE), Pages: 162 - 167, DOI: 10.1109/JCSSE.2015.7219789

[2] Isatou Hydara, Abu Bakar Md Sultan, Hazura Zulzalil, Novia Admodisastro, "**Removing Cross-Site Scripting Vulnerabilities from Web Applications using the OWASP ESAPI Security Guidelines**", 2015 IJST, Indian Journal of Science and Technology, Volume: 8, Issue: 30, DOI: 10.17485/ijst/2015/v8i30/87182.

[3] Li Qian, Zhenyuan Zhu, lun Hu, Shuying Liu, "**Research of SQL Injection Attack and Prevention Technology**", 2015 IEEE, International Conference on Estimation, Detection and Information Fusion (ICEDIF 2015), Pages: 303 - 306, DOI: 10.1109/ICEDIF.2015.7280212

[4] B.Hanmanthu, B.Raghu Ram, Dr.P.Niranjan(2015), "**SQL Injection Attack Prevention Based on Decision Tree Classification**", 2015 IEEE, 9th International Conference on Intelligent Systems and Control (ISCO), Pages: 1 - 5, DOI: 10.1109/ISCO.2015.7282227

[5] Piyush A. Sonewar, Nalini A. Mhetre. "**A Novel Approach for Detection of SQL Injection and Cross Site Scripting Attacks**", IEEE 2015, International Conference on Pervasive Computing (ICPC), Pages: 1 - 5, DOI: 10.1109/PERVASIVE.2015.7087131

[6] Mir Saman Tajbakhsh, lamshid Bagherzadeh. "**A Sound Framework for Dynamic Prevention of Local File Inclusion**", IEEE 2015, International Conference on Information and knowledge technology,Pages:1-6, DOI: 978·14673·7485·9/15

[7] Anastasios Stasinopoulos, Christoforos Ntantogian, Christos Xenakis. "**Commix: Detecting and Blocking command injection flaws.",** DDSUP 2015,Pages:1-9.

[8] https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project, Accessed on: 25/8/2016, 5:45 PM

[9] http://www.veracode.com/security/web-application-vulnerabilities Accessed on: 25/8/2016, 6:30 PM

[10] http://www.informit.com/articles/article.aspx?p=694855&seqNum=2 Accessed on: 25/1/2017, 03:00 PM