CHAPTER 3
SYSTEM ARCHITECTURE

3.1 INTRODUCTION

In WWW, Web based applications are becoming the most dominant way to offer online services. At the same time, web application vulnerabilities are being discovered, which could be compromised by attackers to create an attack in web applications. Most of the information systems are built as enterprise web applications. All these enterprise systems play a vital role in database systems, that provide necessary data as well as store critical information, such as user credentials, financial and transactional information. The vulnerability in web development and mis-configuration of the web server and Database server leads to create an attack on critical information. Traditional security mechanisms like network firewalls, Intrusion Detection Systems (IDS), and use of encryption can protect the network, but they cannot mitigate attacks targeting web applications. The Open Web Application Security Project (OWASP) released its ten most critical web application security vulnerabilities (Christey and Martin 2007) based on the report of the Vulnerability Type Distributions in Common Vulnerabilities and Exposure (CVE). This report ranked XSS as the most critical vulnerability, followed by SQL injection.

XSS is a quite common web application attack technique that involves echoing an attacker supplied code into a user’s browser instance or client via web pages viewed by the target users. Here the attackers host or
inject an attack code written in different static or dynamic contents such as HTML, Java, JavaScript, ActiveX, Flash or any other browser supported technology. A successful XSS attack allows attackers to hijack the user’s account via a cookie, redirecting the user to another website from the one visited, by there facilitating other types of attacks such as phishing or drive-by-download attacks. The XSS attack poses significant risk in cases where the browser interacts closely with file system on the user’s computers for loading content. XSS attacks are commonly used to hijack sessions of users visiting websites facilitating e-Commerce, wherein malicious script/code runs on the user’s client and captures the cookie information of user’s browser, allowing hijacking of the session.

Another common web application attack is the SQLIA. SQL is an interpreted language used in database driven web applications, which construct SQL statements that incorporate user-supplied data or text. If the user supplied data are not properly validated then the user can modify or craft malicious SQL statements, and execute an arbitrary code on the target machine or modify the contents of a database. The SQL injection attack is targeted on a program at the database layer, which is connected to a web application. This SQL injection attack exploits the weakness or vulnerability in the target program to properly verify the input supplied to it through a web form. The Common Weakness Enumeration (CWE) framework provides a unified set of software weaknesses. In CWE, the SQL injection weakness is defined as “not neutralizing or incorrectly neutralizing special elements that could modify the intended SQL command”. In a typical SQL injection attack, the attacker posts specially crafted SQL statements, which are executed in the database server and produce malicious outcomes. The impact and consequences of SQL injection attacks create a major threat to Confidentiality, Integrity, Authentication and Authorization (Johari and Sharma 2012). In a real world scenario, it is very hard to detect the SQL
injection prior to its impact. In most of the scenarios, unauthorized activity is performed by the attacker through valid user credentials, or by using inherent features of the database application, such as malicious modification of the existing SQL Queries of web application that access critical sections of the affected databases.

Similar to the SQL injection attack, XPath injection attacks occur when a web site uses user-supplied information to construct an XPath query for XML data. By sending intentionally malformed information into the web site, an attacker can find out how the XML data is structured, or access data that the attacker may not normally have access to. The attacker may even be able to access privileges on the web site if the XML data is being used for authentication. Today XPath injection is quite common, due to the accelerating use of web services, and use of XML documents instead relational databases or traditional flat files. Hence, XML data is more at risk when it comes to accepting poor input data for XPath parsers.

In order to prevent code injection vulnerabilities, such as SQL injection, XPath injection XSS attack and Session hijacking, a system called “Web Applications Secure System from Code Injection Vulnerabilities through Web Services (WAPS-CIVS)” has been proposed and developed, which will address the code injection vulnerabilities in web applications.

3.2 WAPS-CIVS ARCHITECTURE

Code injection is a type of exploitation caused by processing invalid data input. The concept of injection attacks is to introduce (or "inject") a malicious code into a program, so as to change its course of execution. Such an attack may be performed by adding strings of malicious characters into data values in the form or argument values in the URL.
Figure 3.1 System architecture of WAPS-CIVS
The WAPS-CIVS is the first service based tool to prevent code injection vulnerabilities in enterprise web applications. The WAPS-CIVS consists of major modules, such as the User data interceptor, SQL injection preventer, Session hijacking preventer, Cross-site scripting preventer, XPath injection preventer, Error customizer, and Log file monitor.

### 3.2.1 User Data Interceptor

This module involves intercepting the user inputs of the web form and the header information in the query string from the web applications. Fundamentally, all the code injection vulnerabilities are possible only through the malicious input in the web application. These inputs of the web applications are used to craft malicious queries and scripts to create SQL injection / XPath injection and Cross-Site Scripting attacks. In order to detect and prevent code injection vulnerabilities, every HTTP request and HTTP response have to be intercepted.

In fact, it is common to see other external sources of input, such as HTTP cookie information or server variables like session and application used to build the request query at the end. Once the user provides the input in a client application, these values along with any other dynamically generated values are given to the web application server, which in turn, sends the input parameters to the web server. These input parameters then generate the dynamic query that executes in the database server as well as in the application server (Boyd and Keromytis 2004). If these inputs consist of some malicious values, then the server will change the structure of the query, which will run at the business logic and data logic. Hence, it is necessary to intercept the HTTP request and response of the web applications to validate them, to ensure that the inputs are free from black list characters and malicious values.
In WAPS-CIVS, an Aspect Oriented Programming (AOP) module is designed to detect the vulnerabilities that arise through the dynamically generated query. This module consists of different components that are used for detecting vulnerabilities present in the web application input. Using the AOP to intercept the user input is much better than some of the existing approaches.

### 3.2.2 SQL Injection Preventer

Solutions for SQL injection attacks can be broadly classified as vulnerability identification approaches and attack prevention approaches. The vulnerability identification approach consists of techniques that identify vulnerable locations in a web application that may lead to SQL injection attacks. In order to avoid SQL injection attacks, a programmer often subjects all inputs to input validation and filtering routines, that either detect attempts to inject SQL commands or render the input benign.

In this SQL injection preventer system, the attack prevention approach will be followed by intercepting the SQL query, which will be run at the data logic as shown in Figure 3.2. The user input in the web form will be used by the business logic to craft the SQL query at runtime, which will be run at the data logic to provide the required information to the business logic. If the attacker provides malicious input to the web form, that input will be used only to craft the SQL query which will create the SQL injection in web application. The dynamically generated query can be used to detect any type of SQL injection, since the injection is not only possible through user inputs. The inputs may be from the HTTP header and server variables. The SQL injection preventer consists of the Query Analyzer, Query parser and XML Query format Engine modules, with the XML file and SQL/XML Schema as inputs, to the preventer system. The dynamically generated query will be intercepted and it will be passed to the SQL injection preventer module for
checking its legitimacy, and then the error customization module. The internal modules of SQL injection preventer, analyze the SQL Query and convert the query to XML file for validate with SQL/ XML schema.

![Figure 3.2 Overall view of the SQL injection preventer](image)

The validation results show the legitimacy of the user input, as to whether there are any injection parameters or not. If any error is returned in the validation, then that error information will be passed to the customize error generation module and file log system.

### 3.2.3 XPath Injection Preventer

XPath injection can be prevented in the same way as SQL injection, since XPath injection attacks are much like the SQL injection attacks as shown in Figure 3.3. Common ways to prevent XPath injections are strong input validation and Use of parameterized XPath queries. All user-controllable inputs must be validated and filtered for illegal characters as well as content that can be interpreted in the context of an XPath expression. Parameterization causes the input to be restricted to certain domains, such as strings or integers, and any input outside such domains is considered invalid, and the query fails. The fetched inputs of the user input interceptor are the
inputs for this XPath injection preventer. In WAPS – CIVS, Aspect Oriented Programming (AOP) plays a major role in the detection of XPath injection vulnerabilities, by intercepting the XQuery which was framed by the input parameters.

When the user provides the required inputs to the web forms, they are placed into the XQuery in an appropriate place of the application. Finally, the complete XQuery string is generated for processing the data transaction on XML databases. The generated (or framed) XQuery string with illegitimate inputs may cause XPath injection in a web application. The inputs that lead to XPath injection have to be prevented to run on the XML data. Moreover, the XPath injection is not restricted only by the web form input. It is also possible by the HTTP header or by the cookie (Alvarez and Petrovic 2003). But at the end, all the inputs would be placed into the XQuery to be run on XML data. Hence, the XPath injection preventer system analyzes the XQuery with the help of the XQuery interceptor, XQuery analyzer and XQuery Validator for identifying the vulnerabilities and preventing XPath injection.

3.2.4 Cross-Site Scripting Preventer

XSS attacks leverage insufficient input/output validation in the attacked Web application to inject a JavaScript code, which is then executed
on the victim’s machine within the exploited Web site’s context, thus bypassing the same origin policy. The attacker can craft the injected script in such a way, that it discloses the victim’s confidential information. Since a conceptual solution for the Cross-Site Scripting problem seems infeasible, counter measures currently focus on mitigation techniques to make up for the vulnerabilities still present. There are many strategies like static, dynamic, client-side and server-side techniques, to detect an XSS attack in web applications. But, the ideal solution to prevent an XSS attack will be the server-side technique. Prevention of the XSS attack through a server side solution does not require the user to install any extra components.

In this XSS preventer system, the prevention technique is through the server-side prevention mechanism as shown in Figure 3.4. Since, this prevention system was developed based on the web services, it does not require modifying the business logic of the web application. Here, the intercepted HTTP request and response will be the fundamental parameters to prevent the XSS attack. A very basic module called Script Module Verification will verify the blacklisted characters in the response page against the request page, and encode the vulnerable script present in web applications, by simply displaying it on the web browser.

![Cross Site Scripting Preventer System](image)

**Figure 3.4 Overall representation of cross-site Scripting preventer system**
To check the presence of an XSS script in the request or response, the client request and server response are intercepted, and passed on to the pattern generator module. The pattern generator module generates a pattern like a graph with the help of the collected script present in the web page. For each request and response, this module crawls the page and generates a pattern to check the same origin policy. With the request and the response, there are two patterns, such as the request URL client script pattern and response URL client script pattern. From the two patterns, two adjacency matrices are prepared. By comparing the two adjacency matrices with rows and columns, the XSS prevention module can easily make out whether a malicious script has been added or not. If the dynamically generated response page has any extra vulnerable cross-site scripting code attached to the response page, it will be easily detected. The identified vulnerable script will be encoded with appropriate entity characters, so that, the script will appear in the client browser as it is. The vulnerable script will not be executed in the web browser to allow the attacker to redirect the Web page to a malicious location and hijack the client-server session. As a result, the XSS attack will be prevented with the server-side approach, by means of web services.

3.2.5 Session Hijacking Preventer

Session hijacking is the act of taking control of a user session after successfully obtaining or generating an authentication session ID. Session hijacking involves an attacker, using captured, brute forced or reverse-engineered session IDs to seize control of a legitimate user’s Web application session (Luminitaa 2011), while that session is still in progress.

Since the HTTP protocol does not maintain the session information between the requests, a methodology is needed to maintain the session between every request. In order to maintain the session, for every HTTP request that contains the Session ID (SID) is regarded as belonging to this
particular user. Thus, the SID is a credential, that both identifies and authenticates the user. The protection of this information is therefore essential for the security of the application. This SID could be hacked by the attacker to gain unauthorized access on information or services in a computer system. Therefore, the protection of this session information is essential for the security of the application. In order to prevent the session hijacking attack, a module has been developed in the WAPS - CIVS system, that is shown in Figure 3.5.

![Session hijacking preventer](image)

**Figure 3.5 Session hijacking preventer**

The Session hijacking preventer module consists of the Session ID Fixation preventer module, the XSS Propagation preventer and the Browser Hijacking Preventer. In this approach, a web service based solution is introduced to avoid the change required on both the web browser and the web server. To prevent the Session ID fixation, a technique called dynamic cookie
rewriting and dynamic id generation is used. The dynamic cookie rewriting and dynamic id generation are the part of the web service to prevent session ID fixation. To prevent the browser hijacking, the one-time URL technique is followed. A unique randomized nonce (rnonce) value is generated for each client’s request. As long as the server responds, only to requests for URLs with a valid rnonce value, the attacker is unable to execute the browser hijacking attack. So, to prevent browser hijacking attack, the generated rnonce value is attached to URL, to generate URLRandomization for the one-time URL. Another module in session hijacking prevention is the background XSS propagation module. In this function, a technique called sub-domain tracking is used to find the iframe inclusion or pop under window, which can be misused by the hacker to get the credential information about the user and the session details. These details are further used to hijack the session from the legitimate user.

3.2.6 Error Customizer and Log File Monitor

An error may occur in any web based application. These errors may be exploited by the attacker to create a threat in web application. Whenever a client provides inputs to the web application, they should be legitimate. If these inputs are not legitimate, then an error may be passed to the client browser with some application details. These details help the attacker to create attacks. Usually any application designer will try to trap all the application errors through the Error Handling mechanism. But the exception handling some times uncovers the errors in the web applications. Still some of the uncovered errors could be handled by the web server with the appropriate error code. But the logical errors cannot be handled by error handling or the web server. These logical errors will be disclosed in the system or the database information for the attackers to achieve their intention. To address
this issue, an error customizer module has been developed in the WAPS-CIVS as shown in Figure 3.6.

A log file is a recording of everything that goes in and out of a particular server. The information is frequently recorded chronologically, and is located in the root directory, or occasionally in a secondary folder, depending on how it is set up with the server. The only person who has regular access to the log files of a server is the server administrator, and a log file is generally password protected, so that the server administrator has a record of everyone and everything that wants to see the log files for a specific server.

![Figure 3.6 Error customizer and log file module](image)

The point of a log file monitor is to keep track of what is happening with the server (Krugel et al 2005). If something should malfunction within a complex system, there may be no other way of identifying the problem. If an attacker tries to create an SQL injection or XPath injection or XSS attacks, then the details of the client, i.e., the attacker will be stored in the log file system. Here, the web administrator keeps the web application attacks’ statistics in the server system. This log file monitor module also helps the web site administrator with information about any uncovered threat in web applications. The log file analysis is generally performed by some kind of computer program that makes the log file information more concise and readable. By analysing the log file, the common problems between different systems that might need one major solution to repair them all, may be found.
3.3 WAPS-CIVS with Web Services

A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms, can use web services to exchange data over computer networks like the Internet, in a manner similar to the inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards.

In chapter 2, a detailed discussion is performed with the limitations of the existing approaches. All the approaches have a common limitation, viz, the compatibility issue, and the solution is dependent on the web application. But, by implementing the WAPS-CIVS based on the web services approach, the designed system overcomes the limitations and the overhead is insignificant compared with the assets that depend on the web application.

As the web service is a networked application, it offers many benefits compared to normal distributed applications (Curbera et al 2001). The WAPS-CIVS benefited through the advantages of web services by implementing the entire WAPS-CIVS system with the help of web services. The WAPS-CIVS would prevent SQL injection, Cross-site scripting, XPath injection and Session hijacking. All the prevention systems are developed with the help of web services to overcome the common limitations of the existing approaches. Since, every prevention system is developed through the web service, the entire system has a unique approach to eliminate the compatibility issue and it does not demand any changes in the existing web application code or client browsers in the WWW environment. In addition, loosely coupled components play a vital role in building enterprise web applications. In the WAPS-CIVS, all the preventer systems are loosely
coupled, since they are implemented with the help of web services. Hence, web services technology is a part of the WAPS-CIVS, to prevent Code injection vulnerabilities in web application.

3.4 CHAPTER SUMMARY

The WAPS-CIVS system has been designed for the prevention of the SQL injection attack, XPath injection, detection of Cross-site scripting and session hijacking. For the entire prevention system, the input interceptor, error customizer and log file system are the common modules.

Whenever a user provides inputs to the web application, they will be intercepted by the input interceptor module, and passed to all the prevention systems to detect the presence of any type of code injection vulnerability. If any attack is found in the input of the web application then the WAPS-CIVS would prevent the attack. The client information and the type of attack will be recorded in the log file for further analysis. In addition, a generalized error message will be sent to the client in order to prevent any gain from the error reply. Since, the WAPS-CIVS is developed through web services, the entire system can be deployed independent of to the web application. In addition, the WAPS-CIVS will support for the any platform application, without any compatibility issues.