MJRET
Open Access
Multidisciplinary Journal of Research in Engineering and Technology
Journal homepage: www.mjret.in
ISSN:2348 - 6953

INTRUSION PROTECTION AGAINST SQL INJECTION AND CROSS-SITE SCRIPTING

Rutika Dhapte, Divya Nadanasundaram, Saif Shaikh, Chaitali Mankar,
Prof. Prashant Kumbharkar
Department Of Computer Engineering, Dr. D Y Patil School Of Engineering
Charoli(Bk), Via Lohgaon, Pune, Maharashtra 412105

Abstract - *In this era where the internet is a part of our daily lives, data security in web applications has become very important. Web application vulnerabilities like SQL injection and cross site scripting have become very prevalent. These attacks pose serious risks to confidential database contents. The attackers scope web applications for vulnerabilities and once detected exploit these vulnerabilities to their benefit. The attackers inject SQL statements into the input fields of web application and get access to the application's database. This allows them easy access to sensitive information and lets them modify the database. Another form of attack i.e. XSS attack involves injecting malicious script into a trusted website that executes on a visiting person's browser without their knowledge and thereby making them vulnerable and enables the attacker to access confidential user data, such as session tokens and cookies that are stored on the browser. Reverse Proxy is a technique that we propose to prevent intrusion attacks through input fields by first sanitizing the inputs of web application before being used by them. Once the input is checked and deemed clean it is sent back to the web application for its intended usage. Thus the proxy reduces the burden of the developer to incorporate security measures in the code and helps them focus on creating an application with advanced specifications and leave the issue of checking the inputs for attacks on the proxy server.*

Keywords: *Web vulnerabilities, SQL Attacks, XSS, Reverse Proxy*

1. INTRODUCTION

Many developers don't realize that by leaving their websites vulnerable to security issues in their code they may be making a hacker rich. Consider the example: The French Company

Vupen develops and acquires zero-day vulnerabilities and does not make the vendors aware of it. Instead, they share this information of theirs for their well-paying customers, Forbes reported. At a certain point in 2012, according to X-Force research, SQL injection attacks were the culprit for more than 50% of the data breach where the attack type had been made available. While that number has decreased in 2015, it is still one of the major attack vectors that are in use even now. It is no wonder that SQL injection is at the top of the two most highly recognized lists of software vulnerabilities i.e. OWASP Top 10 and SANS Top 25.Almost every week, we get to hear about a new data breach in the news that reports to us about major companies losing huge amounts of usernames, passwords, credit card numbers, banking transactions i.e. sensitive data, after falling victims to a cyber attacks. According to a recent report by Imperva on attacks on Web Application, SQL Injection (SQLi) saw the biggest rise when being compared to last year with a typical web application suffering 3 times more SQLi attacks. SQLi attacks are very common and stay in the top spot in the OWASP Top 10. It was originally discovered and discussed about publicly during 1998. That is one reason why it can be called an ancient problem. Although SQLi flaws have always been considered to be an easily fixable problem in web application security, they've also been neglected or have gone unnoticed by many web developers. The issue we are dealing with is the rising number of SQL injection attacks. Manual SQLi attacks are time consuming and therefore can lead to scenarios where the attacker intercepts packets and sends it different SQL payload – most hackers prefer automated tools to carry out their SQLi attacks that will scan the application for SQLi vulnerabilities.A person also does not have to be a coder to run these automated tools; it only requires to be given few set of commands to initiate the attack on the target site.It is one of reasons why many "script kiddies" go for automated SQLi tools.XSS vulnerabilities are definitely here to stay and not going away — as many attacks are getting more sophisticated and trickier to detect. The golden age of penetration testing, when penetration testers could readily disclose XSS vulnerabilities just by inserting malicious code into a search box, is over. Now it often takes more skill and effort than that for them to discover the XSS vulnerabilities they need to protection against.In the past several months, it has come to light that Yahoo and Facebook have patched two critical XSS vulnerabilities. It clearly shows that XSS vulnerabilities continue to plague the modern and mature web applications, even for the renowned Internet companies. The XSS vulnerability in Yahoo email was pretty straightforward: the input validation was not robust enough to escape the malicious code, and the attacker was able to break input validation. The one that was patched by Facebook was a little more trickier because it exploited a bug which was in the file upload function to upload malicious JavaScript code and it then invoked the code by calling it from a different application. Also, insufficient input validation XSS vulnerability has been disclosed in the Word Press plug in Ninja Form. Letting third-party

libraries to encode input during the development phase and using a web application firewall in the deployment phase fools web security managers into thinking that their web applications are totally safe from Cross-Site Scripting (XSS) attacks. And while it is a good idea to adopt these techniques, it could prove costly. These protection methods do not actually guarantee that the developed web applications are 100% free of XSS vulnerabilities, and XSS attacks that use more advanced techniques still occur, so when employing third parties care should be taken.There are few types of XSS attacks that are difficult for penetration testers and tools to discover. It is not easy to develop new techniques to fight these, other than enhancing security implementation during the web development and conducting regular security audits to make sure to catch any that were unknowingly included in the code .As we have written attackers use attacks like SQL injection and Cross site scripting which are one of the many attacks to exploit the vulnerabilities of websites and web applications. To learn more in detail about the SQLi refer [2] and for Cross-site scripting attacks refer [32].

2. SQL ATTACKS

An SQL Injection Attack (SQLIA) takes place when an attacker changes the intended effect of a SQL query by inserting new SQL keywords or operators into the input. Now, we define two important characteristics of SQLIAs that for describing attacks: Injection mechanism and Attack intent.

2.1 INJECTION MECHANISMS

It describes the various ways through which we can introduce malicious Sql query into vulnerable web application. Below we explain the most common mechanisms.

1) Injection of malicious statement through user input
2) Injection of malicious statement through cookies
3) Injection of malicious statement through server variables

2.2 ATTACK INTENT

It explains the various intents for which the attacker attacks the application. Some of the attack intents are stated below:

1) Extracting data
2) Adding or modifying data
3) Evading detection
4) Bypassing authentication
5) Executing remote commands
6) Performing privilege escalation

2.3 SQLIA TYPES

Given below are the types of SQL attacks.

1) Tautologies
2) Illegal/Logically Incorrect Queries
3) Union Query
4) Piggy-Backed Queries
5) Stored Procedures
6) Inference
7) Alternate Encodings

Readers who are interested can refer [2] for further in depth explanation on the SQL attacks.

3. CROSS-SITE SCRIPTING

XSS attack can be of three types persistent or non-persistent, or it can be based on a document object model (DOM).

1) Persistent XSS
2) Non persistent XSS
3) DOM-based

Readers who are interested can refer [32] for further in depth explanation on the Cross site scripting attacks.

4. RELATED WORK

SQLrand [1] and Preventing SQL Injection Attacks technique appends random token to SQL keywords in code of the application thus creating random instances of the SQL query language, by randomizing the template query inside the database parser. D-randomizing proxy allows easy installation of their solutions into existing systems, which converts randomized queries to suitable SQL queries. A proxy server then checks that all keywords and operators have the token. If token is not present then attacks can be recognised. The disadvantage is the token can be easily guessed. The grammar that accepts only legal queries are checked at runtime in SQL Guard and SQL Check [8] and [9]. It is used for detection of SQL injection attacks and marking mechanism is used for differentiating suspected input. The structure of the query is checked by SQL Guard before and after the addition of user-input.

In CANDID [5] technique , user input query structure is taken and attacks are guessed by comparing it with the actual query .User given queries are mined by dynamically evaluating

runs over benign candidate inputs. This method is theoretically well established. A very effective tool for detection of SQL injection attacks is CANDID.

AMNESIA [7] combines static analysis and runtime monitoring. In static analysis model is constructed by various queries which can be created by application at database's access point. They intercept the queries before sending them to the database and checking is done. In dynamic phase ,queries that violate the model are not allowed to access the database. The drawback of this technique is it is dependent on its static analysis for building accurate query models.

The required syntactic structure of SQL statements are developed by SQLIDS[11] for security purpose. SQL statements that does not match the specifications are blocked from there execution. Specifications are most important in SQLIDS technique. Specifications are a set of rules that illustrates the expected structure of SQL statement. The original SQL statement is executed in the back end database. The specification is created by the syntactic structure Interception of SQL statements. Filtering is done and the SQL statements produced by the application are not directly sent for execution to the database. Assumption based detection of injected SQL statements id done. Therefore, if the intended structure of the expected SQL commands has been mentioned previously then it becomes easy to detect malicious alterations of this structure.

For detection and prevention of SQL Injection attacks in database using web services, XPATH [3] authentication Technique is used. The login page is redirected to their checking page. In this technique two modules are used i.e. Active Guard and Service Detector. Active Guard, is used as to detect and prevent characters that are suspicious. The validated user input is sent to the Service Detector. SOAP protocol (Simple Object Access Protocol) is used to send the user input to the web service. The user input data is compared with XML_Validator. If the data is matched, the XML_Validator sets a flag value to 1 to Service Detector and valid user can access the web application. If the data is not matched, the flag value is set to 0 and invalid user cannot access the web application. The Service Detector filtration model validates user input from XPATH_validator where the private data is stored. Comparison of user input with data in XPATH_validator is done and if it is same then authorized user is allowed for further processing. Direct access to the database is not allowed by Web Service Oriented XPATH Authentication.

SQLProb [6] is a new approach. In this technique the extracted user input data in syntactic structure of the query can be evaluated. Source code of the application or the database is not required. Also the system can be easily deployed on the any environments without modifications. The SQL proxy-based Block (SQLProb) system has four main components i.e. the query collector, the user input extractor, the parse tree generator, and the user input validator. The query collector is collection phase where all the queries are processed. The

user input extractor implements a global pair wise alignment algorithm to identify user input data. Then the parse tree for the queries is generated. The User Input Validator checks the user input whether it is malicious or not by using the user input validation algorithm. SQLProb is used in two phases i.e. the data collection phase and the query evaluation phase. In the data collection phase, queries are collected by user input validator and are saved. In the query evaluation phase, when a query is given to proxy it is forwarded to the user input extractor and the parse tree generator simultaneously. Reasonable performance overhead is achieved making the system ideal for environments where software or architecture changes are not considered.

A client side technique includes Detecting malicious JavaScript code in Mozilla[13]. Mozilla Firefox web browser is required. All user input is intercepted and logged by the auditing system. Audit system is created for JavaScript code execution for detection of vulnerabilities in browsers like Spider Monkey and Mozilla Firefox. The auditing system continuously checks the implementation of JavaScript . At the time of execution of JavaScript, various function retracts are registered in the JavaScript engine. These are used when a malicious script tries to access specific information that does not exist in the JavaScript engine.

For detection of cross site scripting attack vulnerabilities [14] in the web applications a multi agent scanner is used. Thus multiple agents can operate the system individually. Source code is not required. The system has three modules – web page parser agent, script injector agent and verificator agent. Web page parser agent explores the injection point of stored XSS attack in the web site. The parsing process retrieves information from the web pages it visits and searches the web site following the hyperlinks it finds. There are two different characteristics than the web crawler. First, it follows the hyper-links, targeting the scanned web sites. Second, the information which we get are the forms which are main points for the attacks. Links containing parameters are not stored as entry points as they are not used to insert new information hence they cannot be used for stored XSS attacks. Script injector agent uses collection of web pages discovered by the web page parser agent. In the attack list, different attacks gets listed. The agent injects a set of XSS attack vectors into the various input fields of all the injection points. The next module is verificator agent, it uses the performed attacks list of script injector agent and analyses the application. Input validation errors are checked by this module.

Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis [15] states how XSS attack can be prevented. This technique uses the concept of dynamic data tainting. Dynamic recording of sensitive data is done when scripts are running. When malicious data is being transferred to a third party then avoiding it by logging, preventing the transfer and stopping the program with an error can be done. Dynamic taint and static analysis module are developed for recording the sensitive data in the web browser and intrusion prevention

against cross site scripting attack. When the cookies of user is about to be transferred to a possible attacker, the user can stop the connection. Based on automatic browsing, validation of different technique can be done.

Internet Explorer and Google Chrome have built-in XSS filters. Firefox has a plug-in called Noscript which blocks scripts on all domains. It also uses regular expressions to detect and encode malicious data in URL. IE8's filter [16] uses regular expressions (called heuristics) to identify XSS attack vectors. The HTTP request are first scanned and then checked whether it matches the heuristics or not. If matched, then it creates a signature. The response is scanned to search for scripts that matches with such signatures. For each matching script, a matched character is replaced by another character in order to prevent execution is such script.

Chrome's filter called XSS Auditor [17]mediates between the HTML parser and the JavaScript engine. Only the intercepted part is examined. A script is delivered to the JavaScript engine only if it does not match script found in any input parameter. Chrome filter fails to prevent partial script injection. XSSFilt [18] is similar to XSS Auditor, but has fewer false negatives as it does not check exact string matching .It goes for approximate matching which acts as a drawback.

Stephen W. Boyd and Angelos D. Keromytis [19] in year 2004 proposed an approach based on Instruction Set-Randomization. In this technique, the predefined SQL keywords are changed by adding any integer between them before they are sent to the database. Integers are added in such a way that attackers are not be able to guess but can be recognised by the database. To overcome this problem, a proxy is developed which decodes keywords into original data before giving it to the database server. There is negligible performance overhead found but this method is capable of detecting only tautology type of SQLIA.

Russell A. McClure and Ingolf H. Krüger [20] in year 2005 proposed an approach, based on the concept of Object Oriented Programming. Their solution consists of an executable Sqldomgen which is executed against a database. The output generated by sqldomgen is a Dynamic Link Library (DLL), containing classes which are strongly typed . These classes are called as SQL Domain Object Model (SQL DOM). With the help of this a developer can make dynamic SQL statements without changing any string. Every valid SQL statement is created by object data model and then the schema of the database is generated automatically. Next , the tables and columns contained in the schema are iterated and output is generated. Output developed are the number of files containing a strongly typed instances of the abstract object model. This approach detects the attack in the code at compile time.

William G.J. Halfond and Alessandro Orso [21] in 2005, proposed a technique, which uses a Model-based Approach (AMNESIA).It is used to detect illegal queries before they get execute

on the database. AMNESIA is based on both static and dynamic analysis of queries. Using runtime monitoring static queries are compared with dynamic ones. It identifies the spot in SQL query and then design a SQL query model in order to compute the values of query string given to a database. When Hotspot is reached, then the runtime monitoring starts. If the query is compatible with the model, query gets executed. This method detected 1470 attacks performed for 3500 legitimate accesses to the applications.

Shaukat Ali, Azhar Rauf, Huma Javed [22] gave a method in year 2009. Hash values of username and password for authenticating users to the database is done in this method to protect it against SQLIA. These hash values for username and password are generated automatically when the user enters into database. Authentication of a user is done by his username, password and hash values for username and password. The time overhead of the approach is too small and is 1.3 milliseconds. But this method detected only tautology type SQLIA.

Rattipong Putthacharoen, Pratheep Bunyatnoparat [23] in 2011 uses the method of rewriting the cookies. In this method we have to make the cookies useless for XSS attacks. A proxy agent between user's browser and web server is used, which changes the value of name attribute in the cookies field. The returned cookies are again written at original value before being forwarded to web server. Now even if attackers steal cookies from the database, they cannot be used to get correct user information. The tool detected both categories of XSS attack without having any changes made at the client and server site. But the proxy failed to intercept https requests coming from the client.

SQLGuard [12] and SQLCheck [8] checks the queries at runtime .Taint based approaches like the WebSSARI [24] detects input-validation related errors using information flow analysis. In this approach, static analysis is used to check taint flows. Livshits and Lam [25] use information flow technique to detect when tainted input has been used to construct a SQL query.

Security gateway [26] is a proxy filtering system. It applies input validation rules on the data which is sent to a web application. SQLRand [12] is an approach based on instruction-set randomization. Queries are developed using randomized instruction and not normal SQL keywords. A proxy-filter intercepts queries to the database and keywords are de-randomised.

David Litchfiled [27], introduced techniques to determine the structure of the database application using SQL injection. Chris Anley [28], has stated 'the various ways in which SQL can be 'injected' into the application and noted few data validation and database lockdown issues that are related to this class of attack'. In his other work "(More) Advanced SQL Injections" [28] explains that time delays can be used as transmission channels for accessing data.

Cesar Cerrudo [29], in his white paper demonstrates 'how an attacker could use a SQL Injection vulnerability to retrieve the database content from behind a firewall and penetrate the internal network'. Ofer Maor and Amichai Shulman [30], in their paper, show that SQL Injection could be performed even without any detailed error messages from the server.

Ofer Maor and Amichai Shulman [31] in their paper gave report of all the techniques used to manipulate SQL Injection signatures. Signatures are standard forms in which an SQL manipulation could be done. It is basically done to dynamically change the meaning of the query in the application. They give an overview of all the signatures used to protect the server from SQLIA and its techniques.

5. PROPOSED WORKFLOW

As we have explained earlier many websites suffer from SQL injection and cross site scripting attacks. Not many developers take care to make their code secure and their applications becomes vulnerable. There are developers that prefer to concentrate on developing their applications and not on the security aspects making them an easy target for attackers. Therefore we propose a Reverse Proxy Server which will take care of the security aspect of the web application. The application only has to redirect their user inputs to the Proxy Server before incorporating it into their database. All the inputs to the application are intercepted using a HTML filter and after encryption they are sent to the reverse proxy server for sanitization and to make sure they are clear of any intended attacks. Once the proxy checks it, it will either deem it clean and redirect it to the application for further use or discard it on finding any malicious attacks that might compromise the application. A more detailed description of the modules of the reverse proxy server is given below . Proposed workflow is given shown in the *Fig 1.*

1. SQL Injection Detector
2. Cross-site Scripting Detector.\

The general work of the system is as follows:

1. The client sends the login information to the bank.
2. The request is redirected to the reverse proxy.
3. The filter encrypts the user data using the AES Algorithm.
4. The reverse proxy server decrypts the user data.
5. It performs checks to check whether the data is valid or not.
6. The reverse proxy server gives the access to the account only to the legitimate users.
7. Else the user is denied access.

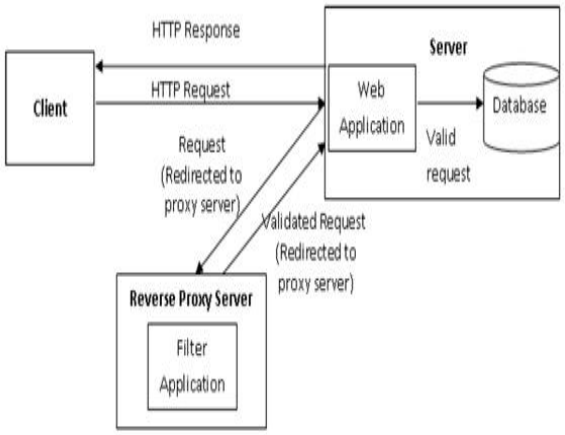
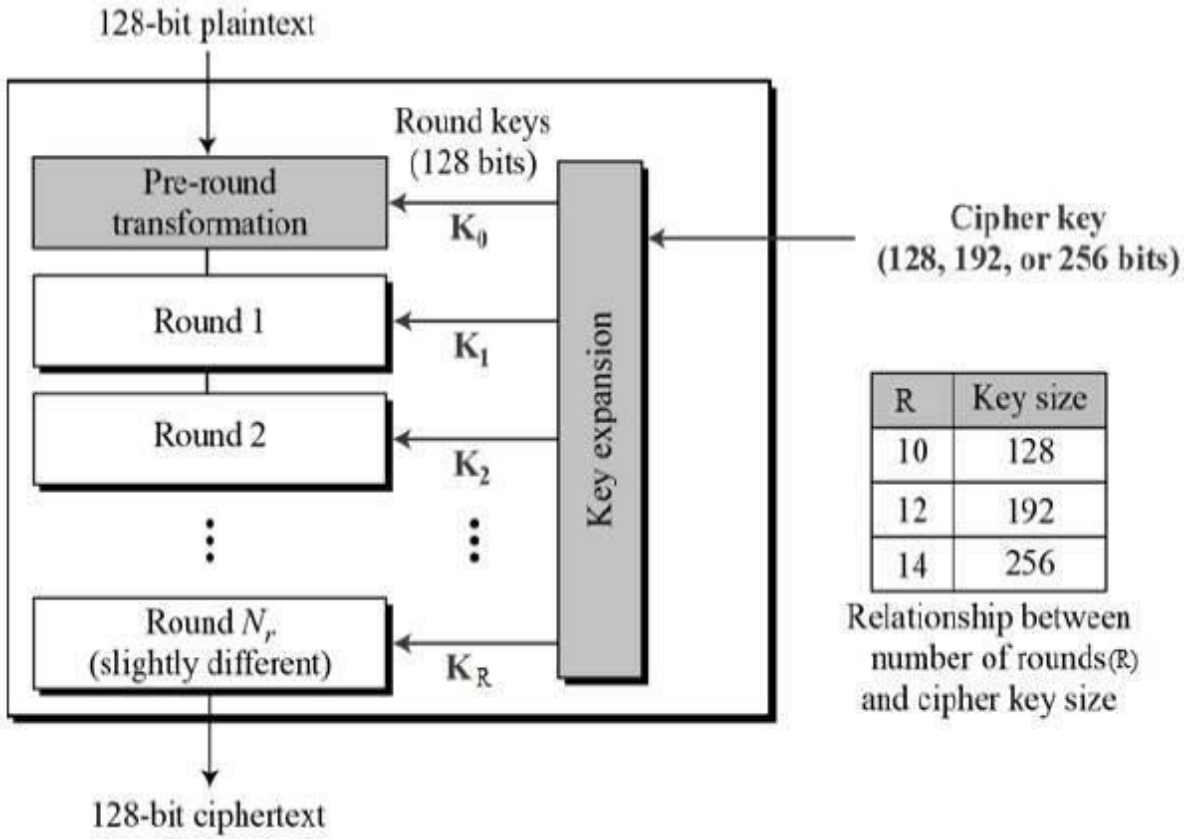


Fig. 1 Proposed Workflow

5.1 AES Algorithm

The popular and widely adopted symmetric encryption algorithm(public key encryption) likely to be encountered nowadays is the Advanced Encryption Standard (AES).

A replacement for DES was needed since the key size for DES is too small.



R	Key size
10	128
12	192
14	256

Fig 2. AES Structure [33]

AES structure is shown in *Fig 2*. The features of AES are as follows -

*Symmetric key symmetric block cipher

*128-bit data, 128/192/256-bit keys

*Stronger and faster than Triple-DES

*Provide full specification and design details

*Software implementable in C and Java

The *Fig 3* describes the algorithm and detail the steps in each round.

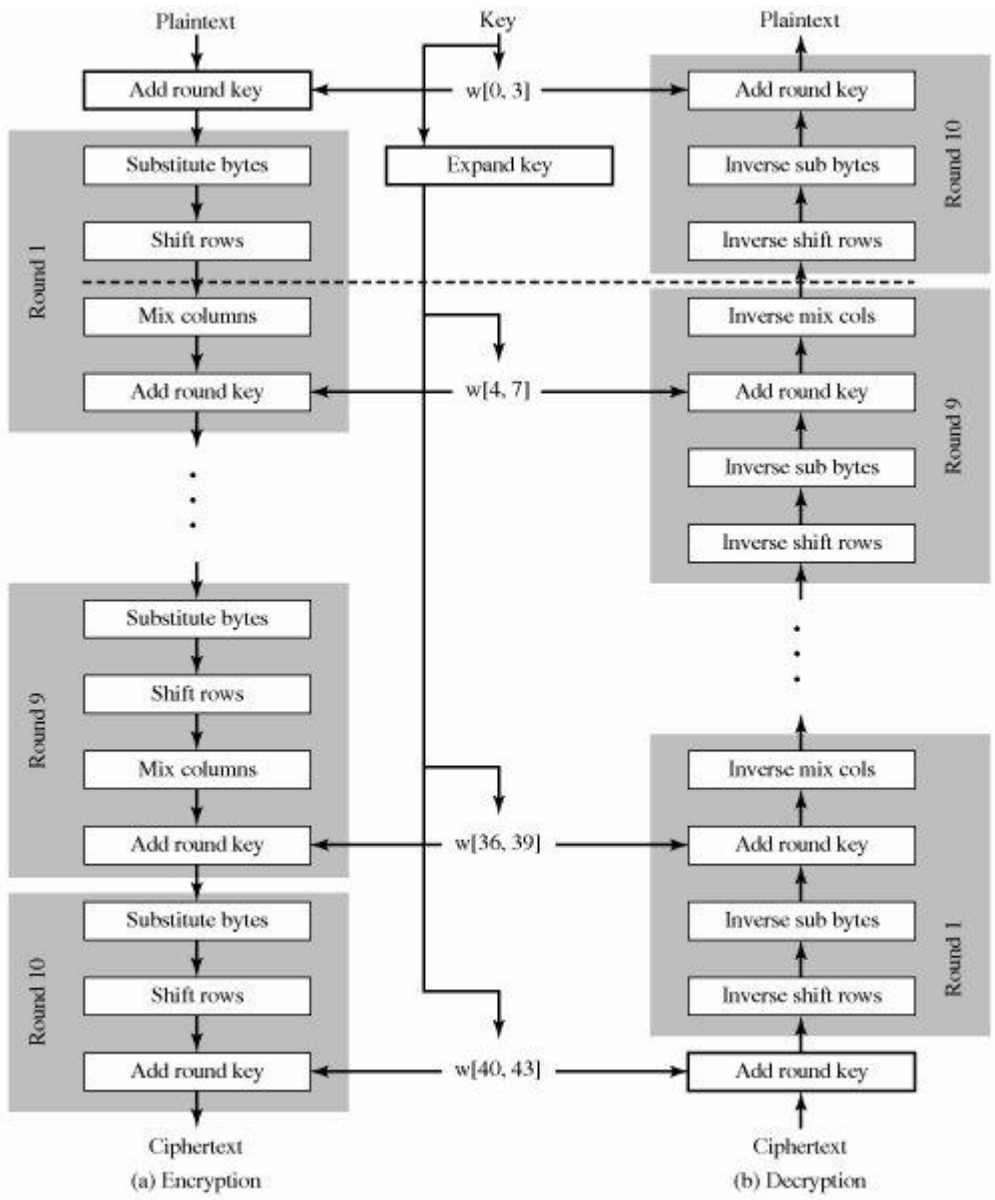


Fig 3. AES Rounds [33]

5.2 Script Detector

Script detector is used to detect the malicious script embedded in the web application. Sanitization process is done which removes all the invalid and unwanted tags, then encodes the remaining input into simple text to avoid any malicious script.

HTML Sanitizer:

HTML Sanitizer removes unsafe tags. It takes a string with HTML code and strips all the tags that do not make part of a list of safe tags. The links href attributes also gets special treatment to remove URLs that trigger JavaScript code execution and line breaks.HTML string in tokens are handled by HTML tokenizers.

Tokenizer:

Tokenizer divides the HTML text within user input into tokens. A token is a single atomic unit of supplied text. As a result of this a list of tokens will be created, and every token in this list is matched with the forbidden tags. Then the HTML Sanitizer forward's the user request to HTML Encoder.

HTML Encoder:

The Html Encode method is useful for displaying text that contain "special" HTML characters such as quotes, angular brackets and other characters by the HTML language. List of some of these special characters and their equivalent encoded value is given below in *Fig 4* which is used by the HTML Encoder to encode the input.

Special Characters	**Equivalent Encoded value**
" {double quote}	&quot;
'{apostrophe / single quote }	&#39;
&	&amp;
<	&lt;
>	&gt;
{space}	
{tab}	

Fig 4. List of special characters and Encoded value

Script Pattern:

This contains all the tags and patterns that are used to match with the tokens which are formed by the tokenizer.

Pattern Matcher:

It just takes the input from the list of tokens and match them with the Script Patterns. All the rejected tags are stored in the invalid tags list and all the accepted tags are forwarded to the HTML Encoder for encoding. *Fig 5* shown below shows the block diagram of Script Detector.

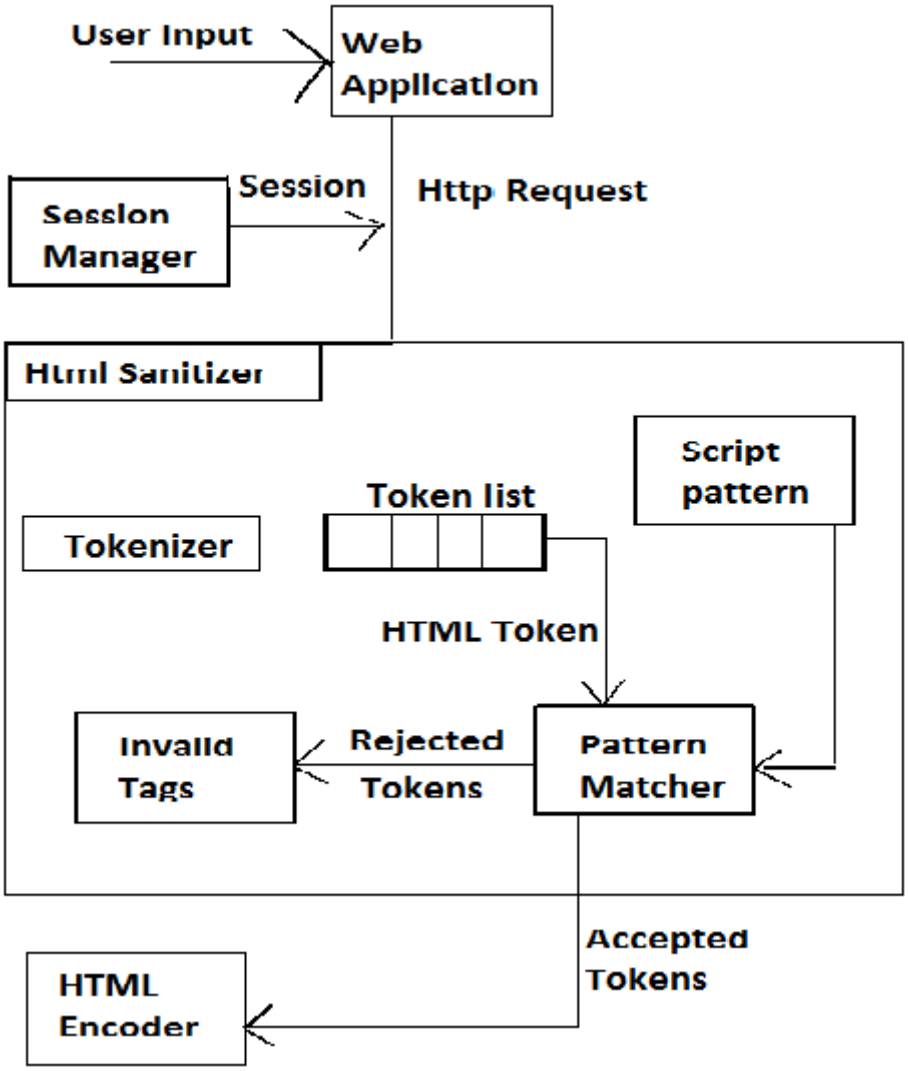


Fig. 5 Block diagram of Script detector

5.3 Query Detector

A Query Detector is a simple tool which is used to test the precision of SQL Queries. It takes request coming from any user and validates the request .

Session Manager:

When HTTP request goes to the web server and the session remains in active until the connection remains active. As soon as the connection is terminated the session terminates accordingly.

Input Valuator:

Any request going on the web server is first validated at the Input_Valuator. It has some stored special characters (e.g. ' - ;) which are often used in writing malicious code for SQL Injection attack. It matches user data in HTTP request with the stored characters. If

no pattern is matched then that request is treated as valid and is forwarded to the next module. *Fig 6* shows the block diagram of Query Detector.

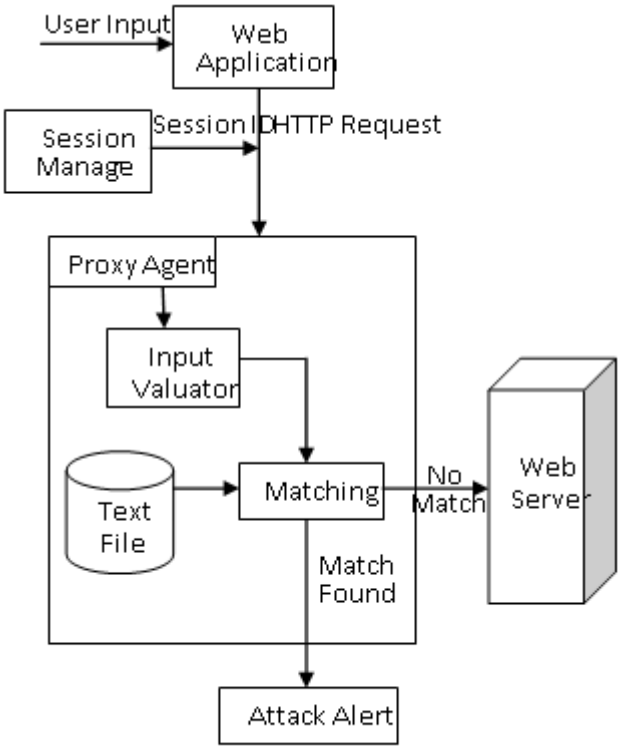


Fig. 6 Block diagram of Query detector.

6. IMPLEMENTATION

We have implemented a system that is the new version of Reverse Proxy Server to prevent attacks like SQL Injection and Cross Site Scripting. The language we have used to develop our web application is java as it is a platform independent language. HTML, JSP, etc are the other technologies that are used. A Banking application is used by us as a web application front. We have implemented 3 modules, which are 1.SQL Injection Detector 2.Cross site Scripting Detector 3.Analysis Module.

Input is sent through the login page of the application by the user. The user input then goes to data redirector program which is on the server and the request sent any the user gets redirected to the reverse proxy server. The request is also encrypted into XML format by the data redirector. The IP address of the system from where the request comes is logged by reverse proxy server. The two modules i.e. SQLInjection preventer and cross site scripting preventer module is installed in reverse proxy server. The request is validated against SQLIA and tokenized by the first module. Signature checks for meta characters, comments and white spaces are done on the request given by the user. If the request is clean then it

goes to the next module. XSS attacks are checked by signature checks through regular expression by the next module. The forbidden tags are prevented by this module.

The attacker's activities are checked by the Analysis module. The IP address of attacker gets blocked if attacker tries to attack more than the specified number of times. Email notification is sent to the user that his account is blocked. The following reports are created by analysis module 1. Attack's List 2. Blocked IP List 3. IP Based Analysis and 4. Web Based Analysis.

The types of attacks, browser, URL, timestamp, attacker's IP address are included in attack list. The IP address of attackers and number of attacks from a specific IP is included in Blocked IP analysis. User-ID, IP address, number of requests and timestamps are checked in IP based analysis. The browser name and attack count is displayed by web based analysis. A table is created based on analysis done which includes 1. Attacker ID 2. Attackers IP address 3. The login name or password used to perform the attack 4. Browser details 5. Timestamp of the attack.

The system architecture of reverse proxy is shown in Fig 6. The request is sent that is redirected to the proxy server by the client to the server. The request is extracted by the sanitizing application and URL is separated from HTTP from SQL statement. Through regular expression URL is sent to signature checker and it is checked. MD5 is used to encrypt the data sent by the user. The validated URL is sent to web application server by the sanitizing application. SQL injection preventer and Cross Site Scripting preventer algorithm is used by the sanitizing application. If URL is found malicious then the request is denied by the server. In case of clean URL, the hashed value is sent to the database of the web application. The user gets account access when user data and stored hash value matches, or else access is not granted. Then the client gets the response and continues with the next request. System architecture is shown in *Fig 7.*

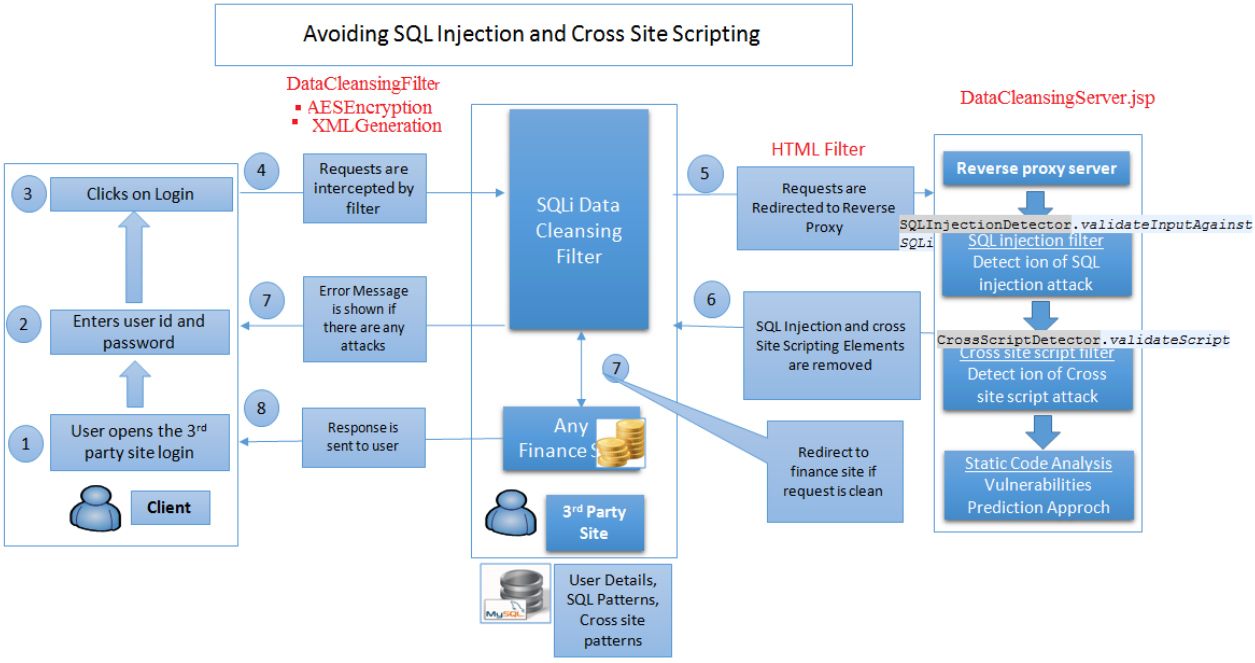


Fig. 7 System Architecture

6.1 Proposed Sanitization Process

SQL Injection preventer: The extracted URL from HTTP and the user data is given as the input. At first we take the user request, then it is parsed and hash table is used to store it and sets its key value. The signature checker uses regular expression to check the parsed request. AND_OR_Integer pattern checks the integer. AND_OR_String pattern checks the string & and/or within single or double quotes. SQL Statement pattern checks the SQL keyword. If the tokenized request matches with it any of these 3 patterns then the request is discarded.

Cross site scripting preventer: The request is taken in form of HTML text. The request is parsed and stored in hash table. The signature checker uses regular expression to check the parsed request. We have a table of forbidden tags and if the parsed request matches the forbidden tags then it is discarded. Else we encode the URL by removing unknown tags.

After all this processing, we get access to the valid account.

6.2 Proposed Sanitization Technology

The configuration is loaded in the system. The request is forwarded to the server by the client. The request is redirected to the reverse proxy and is processed by the sanitizing application. The URL from HTTP and the user data from the SQL statement is extracted by the proxy server. Signature check checks the URL through regular expressions. MD5 hash algorithm is used to encrypt the data. The hashed user data is sent to the server. If the

sanitizing application had marked the URL request malicious then the filter within the server denies the request. Otherwise if the URL is clean then it is sent to database of the application. When the hashed user data and the stored hash value in the database matches then account can be accessed by the user.

6.3 Extraction of User Data

HTTP request is sent to the server by the user and simultaneously from that request SQL statement is extracted. Then the SQL query is tokenized. Comparison of tokenized query is done with all the SQL queries in the database. The tokenized queries are transformed into XML format. Pattern matching algorithm is used. The analysis of the query is done and the query is divided in small parts. The prototype document stores and maintains the query pertained to the next particular application. For example, query is, SELECT * FROM workers WHERE login='admin' AND password= 'XYZ' OR '1=1'. This query is converted into XML format using a XML schema.

6.4 Signature check

The signature check uses regular expression and checks for the SQL Injection attacks. URL from HTTP request is extracted and then tokenized. Small parts of the URL are checked by the regular expression. If any SQL Injection signature is found matching then request is considered malicious. The user does not get access to the account.

7. EXPERIMENTAL ANALYSIS

The system that we have implemented protects the Banking Apllication from SQL injection and cross-site script attacks and also keeps a record of all the attacks that has taken place. The data on the attacks are available to admin of the Banking Web application. The login page is shown in the Fig 8.

Fig 8. Admin login

The data on the attacks can be views under the various options shown in the Figure 9. The Attack_list shows all the attacks that have ever been made to the system and have been stopped. It shows the field in which the attack originated, the account that was used by the attacker, his browser, his ip address and the kind of pattern with which the attack matched with from our database.

	Sr.No	Field Value	AttackField	Browser	Pattern
2	1	'# drop user #'	'password'	'Mozilla/5.0 (Wi	3
3	2	'# drop user #'	'password'	'Mozilla/5.0 (Wi	2
4	3	'# insert into user #'	'password'	'Mozilla/5.0 (Wi	3
5	4	'# insert into user #'	'password'	'Mozilla/5.0 (Wi	2
6	5	'12; drop useraccounts--'	'password'	'Mozilla/5.0 (Wi	3
7	6	'1=1 OR'	'password'	'Mozilla/5.0 (Wi	101
8	7	'831 #select * table#'	'password'	'Mozilla/5.0 (Wi	3
9	8	'891 'and 1=1''	'password'	'Mozilla/5.0 (Wi	1
10	9	'891 'and 1=1''	'password'	'Mozilla/5.0 (Wi	2
11	10	'891 'or 1=1''	'password'	'Mozilla/5.0 (Wi	1
12	11	'891 'or 1=1''	'password'	'Mozilla/5.0 (Wi	2
13	12	'891 'or 1=1''	'password'	'Mozilla/5.0 (Wi	3
14	13	'891 'or 1=1''	'password'	'Mozilla/5.0 (Wi	1
15	14	'891 'or 1=1''	'password'	'Mozilla/5.0 (Wi	1
16	15	'891 'or 1=1''	'password'	'Mozilla/5.0 (Wi	2
17	16	'891 'or 1=1''	'password'	'Mozilla/5.0 (Wi	2

Fig 9. Attack list

The ip_base_analysis shows only the ip addresses through which the attacks have originated. A pie chart has also been provided to make it easier to visualize. It is shown in the Figure 10 given below.

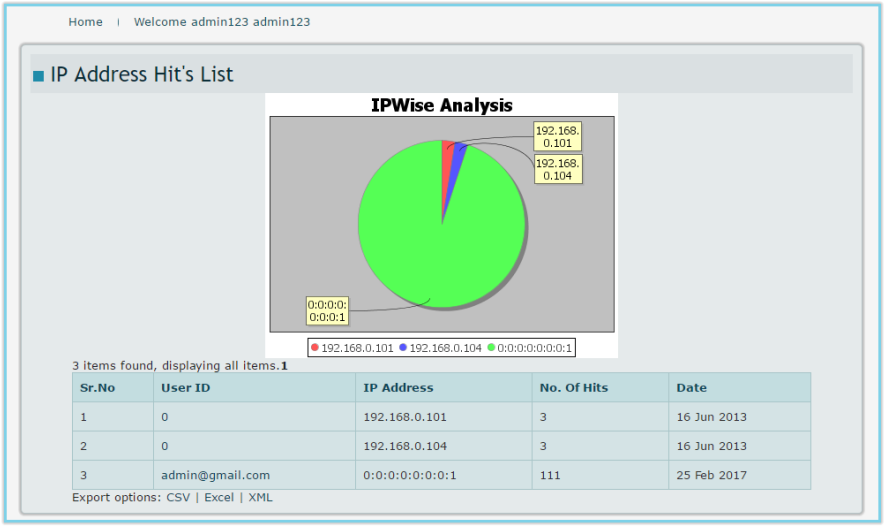


Fig 10. IP Base Analysis

The web_base analysis stores how many times an attack has originated from a particular browser. It too has a pie chart to depict it. View the Figure 11 given below.

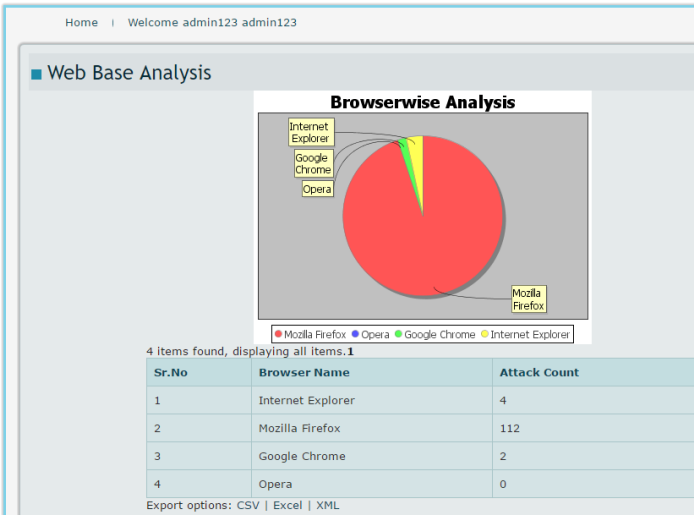


Fig 11. Web Base Analysis

The blocked_ip list serves the purpose of storing the ip addresses that can no longer access the Banking application. If an atttack originates from the same ip address equal to thirteen times, then the ip address is stores in blocked ip list. Also the account that was used to login during these times, the account user of that account receives a mail shown in Fig 12 from the admin informing him about the attacks and the blocked status. Once the ip is blocked, if the ip is used again the web application shows a message that this ip is blocked.

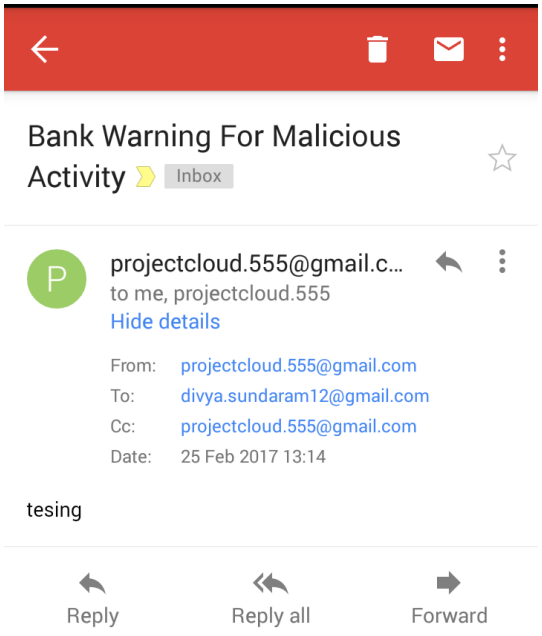


Fig 12. Email Message

Using these unique lists we can view and analyze the attacks that take place on the web application.

8. CONCUSION

In this paper, we have discussed various methods for detection and prevention of SQL injection attacks and Cross site scripting attacks in brief. The goal of the proposed Reverse Proxy is to handle the security issues of a web application. The reverse proxy framework for intrusion prevention uses sanitization technique for discovering SQLIA and XSS attacks. The system will be very effective in detecting and preventing the attacks from intruding the web application. With the help of SQL injection preventer module and Cross site scripting preventer module, we can protect web application against the SQLI and XSS attacks. Reverse proxy prevents the malicious script without making any changes to the source code of the web application. It uses AES encryption while sending the confidential user data to the reverse proxy server, thereby keeping the sensitive information safe from hackers.

ACKNOWLEGEMENT

We would like to express our gratitude to our internal project guide Prof. Prashant Kumbharkar (Head Of Department) for giving us an opportunity and providing us valuable guidance in developing and putting together this for intrusion Protection system.

REFERENCES

[1] Stephen W. Boyd and Angelos D. Keromytis, "SQLrand: Preventing SQL Injection Attacks," Applied Cryptography and Network Security (ACNS) Conference, LNCS 3089, 2004

[2] W. G. J. Halfond, et al., "A Classification of SQLInjection Attacks and Countermeasures," IEEE International Symposium on Secure Software Engineering, Arlington, VA, USA, 2006

[3] IndraniBalasundaram 1 Dr. E. Ramaraj, "An Approach to Detect and Prevent SQL Injection Attacks in Database Using Web Service", IJCSNS International Journal of Computer Science and Network Security, January, 2011.

[4] ShaimaaEzzatSalama, Mohamed I. Marie, Laila M. El-Fangary&Yehia K. Helmy, "Web Anomaly Misuse Intrusion Detection Framework for SQL Injection Detection", International Journal of Advanced Computer Science and Applications,2012.

[5] VeeraVenkateswaramma P, "An Effective Approach for Protecting Web from SQL Injection Attacks", International Journal of Scientific & Engineering Research, Volume 3, 2012.

[6] Anyi Liu, Yi Yuan, DumindaWijesekera, "SQLProb: A Proxy-based Architecture towards Preventing SQL Injection Attacks", SAC'09, Honolulu, Hawaii, U.S.A, 2009.

[7] William G.J. Halfond and Alessandro Orso "Preventing SQL Injection Attacks Using AMNESIA," ICSE'06, Shanghai, China, 2006.

[8] Zhendong Su, Gary Wassermann," The Essence of Command Injection Attacks in Web Applications", Charleston, South Carolina, USA, 2006.

[9] Gregory T. Buehrer, Bruce W. Weide, and Paolo A. G. Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks", Lisbon, Portugal, 2005.

[10] Anil Kumar Mandapati, Adilakshmi. Yannam, "Web Application Vulnerability Analysis Using Robust SQL Injection Attacks", International Journal of Engineering Trends and Technology, 2012.

[11] KonstantinosKemalis and TheodorosTzouramanis, "SQL-IDS: A Specification-based Approach for SQL-Injection Detection", Fortaleza, Ceará, Brazil, 2008.

[12] Gregory T. Buehrer, Bruce W. Weide, and Paolo A. G. Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks", Lisbon, Portugal, September-2005.

[13] OysteinHallarakar and Giovanni Vigna, "Detecting Malicious JavaScript code in Mozilla", IEEE International Conference on Engineering of Complex Computing System(ICECCS), 2005.

[14] E. Galan, A. Alcaide, A. Orfila, J. Blasco, "A Multi Agent Scanner to Detect Stored XSS Vulnerabilities," IEEE, ICITST, 2010.

[15] P. Vogt, F. Nentwich, N. Jovanovic, C. Kruegel, E. Kirda, G. Vigna, "Cross Site Scripting Prevention with Dynamic Data Tainting and Static Analysis," Network and Distributed System Security Symposium, 2007.

[16] David Ross. IE 8 XSS Filter Architecture/Implementation. blogs.technet.com/srd/archive/2008/08/19/ie-8-xss-filter-architectureimplementation.aspx.

[17] Daniel Bates, Adam Barth, and Collin Jackson. "Regular expressions considered harmful in client-side XSS filters" WWW"10, Raleigh, North Carolina USA, 2010

[18] R. Pelizzi and R. Sekar "Protection, usability and improvements in reflected XSS filters" Proc. of the 7th ACM Symposium on Information, Computer and Communication security, ASIACCS"12, Seoul, Korea 2012.

[19] S. W. Boyd and A. D. Keromytis. SQLrand: Preventing SQL Injection Attacks. In Proceedings of the 2nd Applied Cryptography and Network Security Conference, pp 292–302, June 2004

[20] R.A. McClure, and I.H. Kruger, "SQL DOM: compile time checking of dynamic SQL statements," Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on, pp. 88- 96, 15-21 May 2005.

[21] William G.J. Halfond, Alessandro Orso, "AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks", ACM-05 USA, November 7-11, 2005, pp 174183 Long Beach, California.

[22] S. Ali, SK. Shahzad and H. Javed, "SQLIPA: An Authentication Mechanism against SQL Injection," European Journal of Scientific Research ISSN 1450216X Vol.38 No.4 (2009), pp 604-611.

[23] Rattipong Putthacharoen, Pratheep Bunyatnoparat, "Protecting Cookies from Cross Site Script Attacks Using Dynamic Cookies Rewriting Technique", ICACT 2011 pp 1090-1094

[24] Y.Huang, F.Yu, C. Hang,C.H. Tsai, D.T.Lee and S.Y.Kuo, (2004) "Securing Web Application Code by Static Analysis and Runtime Protection", Proc. International World Wide Web Conference '04, pp. 40-52.

[25] V.B. Livshits and M.S. Lam, (2005) "Finding Security Errors in Java Programs with Static Analysis", Proc. Usenix Security Symposium '05, pp. 271-286.

[26] D.Scott and R.Sharps, (2002) "Abstracting Application-level Web Security", Proc. International Conference on the World Wide Web '02, pp. 396-407.

[27] David Litchfield, (2001) "Web Application Disassembly with ODBC Error Messages," Blackhat group, White Paper.

[28] Chris Anley, (2002) "Advanced SQL injection in SQL server applications", Next Generation Security software Ltd., White Paper.

[29] Cesar Cerrudo, (2002) "Manipulating Microsoft SQL Server Using SQL Injection", Application Security Inc., White Paper.

[30] Ofer Maor and Amichai Shulman, (2002) "Blindfolded SQL injection", Imperva Inc., White paper.

[31] Ofer Maor and Amichai Shulman, (2003) "SQL injection signature evasion", Imperva Inc., White paper.

[32]Imran Yusof, Al-Sakib Khan Pathan,"Mitigating Cross-Site Scripting Attacks with a Content Security Policy," Computer (Volume: 49, Issue: 3, Mar. 2016)

[33] https://www.tutorialspoint.com/cryptography/advanced_encryption_standard.htm