ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 11, November 2014

IMPLIMENTATION APPRAOCH FOR SECURE WEB APPLICATION BY DIFFERENT PREVENTION STRATERGIES

Vibhakti Mate[1], Milind Tote[2]

Computer Science & Engineering Department, Nuva College of engineering & technology, Nagpur, India [1,2]

Abstract: The World Wide Web (WWW) is delivering a broad range of sophisticated web applications for business, net banking, online shopping etc., However, many web applications go through fast development phases with short time, making it difficult to eliminate vulnerabilities. This kind of Web applications are target of hackers. From an application point of view, vulnerability identification are critical and often over looked as a source of risk. OWASP develop tools and best practices to support developers in the development and operation of secure web applications. According to OWASP, Web applications vulnerable to attacks such as SQL injection and Cross-Site Scripting, Cross Site Request Forgery, Broken Authentication and Session management as mention below. In this paper we focus mainly on some of the security risks listed by OWASP top ten, such as Cross-site scripting, Cross-site request forgery, Authorization, as well as client side Validations. For the realization of this paper, the new security framework helps to make the applications more secure against these risks. In this paper we are implementing identification of vulnerabilities of web application i.e., SQLI, CSRF, XSS and Broken Authentication and session management, Insecure Cryptographic Storage, Insecure Direct Object References, Failure to Restrict URL Access, Unvalidated Redirects and Forwards etc., to find out their prevention strategies over existing web application. The main objective of this paper is to create a secure web application that provide security when user is login or while user is logged on. Web application must be secure from the attacks that are listed in above section and show how these attacks are used to compromise user identity and credentials. In this paper we are proposing a framework for building secure and anti-theft web applications that must be secure from above listed attacks by improving existing web prevention techniques.

Keywords: Vulnerabilities, SQL Injection attack, Cross Site Request Forgery, Cross Site Scripting, Broken Authentication and Session management, Open Web Application Security Project.

1. INTRODUCTION

The popularity of web applications has increased day by day, mainly because of its client-server architecture and its accessibility from all over the world. They are used in various different environments. As their use in the critical-environment increases, the sophisticated attacks against these applications also have increased and securing applications against these attacks have become very important. In order to develop high quality dynamic web applications, the developer uses various Web based application frameworks. It reduces the burden in the software development life cycle and still, the most important aspect that needs to be considered is, how to provide security to all these applications. We provide importance of security features in the web based application framework by the following statements:

Consideration of security in the System Development Life Cycle is essential to implementing and integrating a comprehensive strategy for managing risk for all information technology assets in an organization.

The integration of security in the software development life cycle of web application, however, still requires a developer to possess a deep understanding of security vulnerabilities and attacks.

Web application security must be addressed across the tiers and at multiple layers. A weakness in any tier or layer makes your application vulnerable to attack.

The main objective of this project is to create a web application that provide security when user is login and while user is logged on. Web application must be secured from the attacks that are listed below and show how these attacks are used to compromise user identity and credentials.

Attacks on Web Application

Most of the attacks on web application are performed to steal confidential data or to deface website or stealing session cookies. And reason for this attacks are some vulnerabilities in web application coding and in application developing methods. This vulnerabilities from which on some of very severe damaging attack can be perform. So while developing application itself developer can avoid that vulnerability.

Type of Attacks

There are several Web application attacks that can be used to take control over application or deface application or stealing confidential data. According to OWASP(Open Web Application Security Project) there are 8 attacks that are most severe and dangerous for any web application. That attacks are as follows[1][15]:

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 11, November 2014

1. SQL Injection Attack
2. Cross-Site Scripting (XSS)
3. Broken Authentication and Session Management
4. Insecure Direct Object References
5. Cross-Site Request Forgery (CSRF)
6. Insecure Cryptographic Storage
7. Failure to Restrict URL Access
8. Unvalidated Redirects and Forwards

2. WEB APPLICATION VULNERABILITIES

Here we are mentioned eight different attacks on web applications, to show how we can prevent from these to achieve secure web application.

2.1 SQL-Injection(SQLI):

An SQL injection attack is a type of attack on web applications that exploits the fact that input provided by web clients is directly included in the dynamically generated SQL statement. SQLIA is one of the foremost threats to web applications. Injection flaws as SQL injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.SQL injection is a technique for maliciously exploiting applications that use client-supplied data in SQL statements. Using SQLIAs, an attacker may be able to read, modify, or even delete query to its underlying database, the attacker's embedded commands are executed by the database and the attack succeeds. The results of these attacks are often dangerous and can range from leaking of sensitive data to the destruction of database contents.

SQL Injection Types:

These are the classification of SQL injection types according to Halfond, Viegas and Orso researches [2,5].

1. Tautology
2. Logically incorrect queries
3. Union queries
4. Piggy-backed Queries
5. Stored Procedure
6. Blind Injection
7. Timing Attacks

Prevention Strategy:

We propose a newly automated approach for dynamic detection and prevention of SQLIA. Our approach works by identifying "trusted" strings in an application and allowing only these trusted strings to be used to create the semantically relevant parts of a SQL query such as keywords or operators. The general mechanism that we use to implement this approach is based on Regular Expressions, which marks and tracks certain data in a program at runtime. The developer sometimes tests the application by using the Code analysis tool. It traces the data flow through the application and avoids dynamic queries approach as well as checks the interpreter. It is also recommended to use object relational mapping tools such as hibernate etc. that verifies the input data on the developer's behalf. In this, the application uses unverified data to form the above vulnerable SQL query. The attacker further modifies the id parameter to **'or '1'='1.** This modified query means that, it has to return all the records from the account tables, instead of returning to only single records. This weakness discloses the database's table information, the complete takeover of the database, and possibly even the server hosting the database.

2.2.Cross-Site scripting (XSS):

Cross-site scripting is an attack against web applications in which scripting code is injected into the output of an application that is then sent to a user's web browser. In the browser, this scripting code is executed and used to transfer sensitive data to a third party i.e., the attacker. Currently, most approaches attempt to prevent XSS on the server side by inspecting and modifying the data that is ex-changed between the web application and the user. Unfortunately, it is often the case that vulnerable applications are not fixed for a considerable amount of time, leaving the users vulnerable to attacks. The solution presented in this paper stops XSS attacks on the client side by tracking the flow of sensitive information inside the web browser. If sensitive information is about to be transferred to a third party, the user can decide if this should be permitted or not. As a result, the user has an additional protection layer when surfing the web, without depending on the security of the web application[6][9].

Prevention Strategy:

Cross-site scripting is one of the most frequent vulnerabilities found in modern web applications. Nevertheless, many service providers are either not willing or not able to provide sufficient protection to their users. This paper proposes a novel, client-side solution to this problem. By modifying the popular Firefox web browser, we are able to dynamically track the flow of sensitive values e.g., user cookies on the client side.

Whenever such a sensitive value is about to be transferred to a third party, the user is given the possibility to stop the connection. To ensure protection against more subtle types of XSS attacks that try to leak information through non dynamic control dependencies, we additionally employ an auxiliary, efficient static analysis, where necessary. With this combination of dynamic and static techniques [3,5], we are able to protect the user against XSS attacks in a reliable and efficient way.

To validate our concepts, The input validation is a preferred approach for handling the entrusted data All the special characters first need to be verified and encoded before placing them into the output. Otherwise, security mechanisms can bypass the injected code inside the documents that were later stored in the web application.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 11, November 2014

2.3 Cross site request forgery (CSRF):

CSRF is a kind of attack which forces an end user to execute unwanted action on a web application in which they were currently authenticated. With a little help of social engineering like sending a link via email/chat, an attacker may forces the user of a web application to execute actions of the attacker's choosing. CSRF vulnerabilities are very common, and consequences of such attacks are most serious with financial web- sites. We propose Browser-Enforced Authenticity Protection (BEAP), a browser-based mechanism to defend against CSRF attacks[10]. BEAP infers whether a request reflects the user's intention and whether an authentication token is sensitive, and strips sensitive authentication tokens from any request that may not reflects the user's intention. The inference is based on the information about the request and heuristics derived from analyzing real-world web applications.

Prevention Strategy:

The most popular suggestion to prevent CSRF attack is to append non predictable challenging token with each user's request. This happens when the user requests a page from the server. The server first creates a session instance or extracts the existing session object for that user from the maintained session pool. It further generates long and secure hash based a random token by using a significantly secure hashed algorithm, such as sha-256 etc. Then it associates the newly generated random token as a hidden text field within the session and responds back to the browser. The browser stores the session cookie inside the cache and places the random token as a hidden field inside the web page. The server receives the hidden random token and a piece of the session on each subsequent request. Further, it verifies that the session value and hidden random token are the same as stored in the session maintained for that user at the server's side. If they are not the same then the server responds back with an error message; otherwise, it generates again a new random token and follows the same procedure, as described above. In addition, it is important to consider some of the points during the generation and maintenance of the token. The size of the generated token should be immensely long, secure, and hard to predict by the attacker; otherwise, the attacker is able to authenticate himself to the server as a valid user with a random token and session id. However, it is possible that the XSS flaw can also grab the session token.

2.4 Broken authentication and Session Management:

Web applications include and cover a number of services such as commercial transactions and mail exchange. The deployment of applications via web has many benefits, for instance (1) it increases the reach of application owners to the intended users, and (2) reduces the maintenance and deployment costs. The wide acceptance and usability of web applications however has brought them into the focus of cyber attacks of various sorts. Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users' identities[7][8].

Prevention Strategy:

In this paper for Session management ,to improve security we suggest the use of cookies, enforcement of the same origin policy for the cookies, and usage of SSL for any traffic comprising of session IDs and credentials. Technical solutions towards the security of web applications from broken authentication problems are the usage of S-HTTP during authentication and cryptographically protecting user credentials i.e., for instance by using hashing or encryption. These solutions protect authentication data but the solution creates communication overhead and need further optimization[8].

2.5 Failure to Restrict URL Access

This attack is also called **'forced browsing'**, in which a brute force method is used to find unprotected pages in the web application and to access URL links based on specific information. This risk is indeed as simple as it sounds; the user is able to access the resources, though they don't have enough rights to happens because of the complex security model used inside a project and the project which is sometimes difficult for security specialists and developers to understand. If the complexity of the project increases, the probability of the error also grows and some pages will be missed out. Sometimes, the **'hidden'** or **'special'** URL is rendered to the administrator and the special users in the presentation layer[15].

Prevention Strategy:

The security experts or the developers need to plan authorization by creating a security matrix that maps the roles to the functions of the application. The Web application not only provides access control to the URL, but also confirms to the business logic residing in the application. As it generally happened that the access control is placed into the presentation layer, but it leaves the business layer unprotected. Moreover, It is also not sufficient to ensure only once during the process that the user is authorized to resources and then leaves it unchecked during the subsequent steps. Otherwise, the attacker may skip the steps of authorization and forge the parameter value necessary to continue on the next steps. One should assume that the users might be aware of the special or hidden URLs or API and provide protection against[14].

2.6 Insecure Direct Object References

The Insecure direct object references vulnerability takes place when the web application exposes references of an

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 11, November 2014

internal implementation of an object, such as a file, URL, directory, or database key to the users. The attacker can modify the internal implementation of the object in order to gain access control on it. The **'open redirect'** and **'open directory'** two categories of the vulnerabilities. In the case of the **'open redirect'**, the user's request is redirected to the same or a different web application based on the parameters that have been passed with the URL. The **'directory traversal'** renders the important files or directory information that is stored in the local machine where the application is running.

Prevention Strategy:

The best way to protect the application against direct object reference attacks is through the validation of private object references. Others include the avoidance of the exposition of private object references to the users[14].

2.7 Insecure Cryptographic Storage

Many web applications do not properly protect sensitive data, such as credit cards, SSNs, and authentication credentials, with appropriate encryption or hashing. Attackers may steal or modify such weakly protected data to conduct identity theft, credit card fraud, or other crimes. In this type Data and Credential are rarely protected with cryptographic functions because of that Data collected can be used by attackers i.e. Crimes like Credit Card Fraud. Mostly common problems aries[14]:

• Not encrypting sensitive data
• Using home grown algorithms
• Insecure use of strong algorithms
• Continued use of proven weak algorithms (MD5, SHA-1, RC3, RC4, etc.)
• Hard coding keys, and storing keys in unprotected stores

Prevention Strategy:

Should be protected with cryptographic tools
1. Encryption
• If you need to read and write data: symmetric encryption (e.g. DES, AES)
• If reading and writing are done by different entities: asymmetric encryption (e.g. RSA)
2. One-way hash functions
• One input has always the same output
• Impossible to go from the output back to the input
• No collision can be generated (two inputs having the same output)
• Example : SHA-256

2.8 Unvalidated Redirects and Forwards

Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.If user is sumbitting a request to your webpage, Attacker links to unvalidated redirect and tricks victims into clicking it. Link is to a valid site, so the users are more likely to click on it[14].

Prevention Strategy:

Attempts to install malware or trick victims into disclosing passwords or other sensitive information. Identify target URL with all the parameters. URL should contain only allowed and needed parameters. Avoid redirects and forwards. Don't involve user calculating the destination address. If you cannot avoid this, check if the supplied value is valid and authorized for the user.

3. ENHANCEMENT

Initially, websites were static and the interactions between users and web servers were very limited. The implementation of dynamic websites through server side scripts made it possible to dynamically generate web pages for interactions between users and web servers. These advancements in web applications deploy the occurrence of critical attacks as mentioned by OWASP. This applications built an architecture, indicate to develop Secure Software Engineering practices to achieve vulnerabilities free web application. Here we are mentioned eight different attacks on web applications, to show how we can prevent from these to achieve secure web application.

We propose a new highly automated approach for dynamic detection and prevention of SQLIAs. Intuitively, our approach works by identifying "trusted" strings in an application and allowing only these trusted strings to be used to create the semantically relevant parts of a SQL query such as keywords or operators. The general mechanism that we use to implement this approach is based on Regular Expressions, which marks and tracks certain data in a program at runtime.

In this project we have developed a highly automated approach for protecting Web applications from SQLIAs. This application consists of 1) Using Regular expressions to track trusted data at runtime, and 2) Allowing only trusted data to form the semantically relevant parts of queries such as SQL keywords and operators. 3) Performs syntax-aware evaluation of a query string immediately before the string is sent to the database to be executed. The project also provides practical advantages over the many existing techniques whose application requires customized and complex runtime environments: It is defined at the application level, requires no modification of the runtime system, and imposes a low execution overhead.

4. IMPLEMENTATION

With all this theoretical background about basic security issues, it is time to make something useful out of it. The beginning of this section describes the architecture of the newly developed security framework which is describe below. Implementation of this paper turns the theoretical design into a working system. Thus it can be considered to

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 11, November 2014

be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

4.1. Creating secure login script for web application.
4.2. Prevent SQL injection and Cross Site scripting in web application.
4.3. Secure Session and Token generation for preventing CSRF attack.
4.4. One time session cookies and use SSL for encrypted communication.
4.5. Validation of Private Object Reference
4.6. Encrypt password using SHA-256 algorithm
4.7. Creating Security Matrix to Restrict URL

Implementation Description

4.1. Creating secure login script

In each and every web application login is the first and most important step to authenticate user. So most of the attacks also performed on login page and that is the reason why login must be secure and unique.
For creating secure login script username and most important password must be stored in database with in hashed form and if it is stored with random salt then if hacker come to know about password then also he can't regenerate password.
For more security immediately after login session must be start with cookies that will be set until user is logged in.

4.2.Prevent SQL injection and Cross Site scripting

SQL injection and Cross Site scripting attacks are most dangerous attacks on any web application so prevention from these attacks are most important.
Input validation is a challenging issue and the primary burden of a solution falls on application developers. However, proper input validation is one of your strongest measures of defense against today's application attacks. Proper input validation is an effective countermeasure that can help prevent XSS, SQL injection, buffer overflows, and other input attacks.
Input validation is challenging because there is not a single answer for what constitutes valid input across applications or even within applications. Likewise, there is no single definition of malicious input. Adding to this difficulty is that what your application does with this input influences the risk of exploit.
The following practices improve your Web application's input validation:
Assume all input is malicious.
Centralize your approach.
Do not rely on client-side validation.
Constrain, reject, and sanitize your input.
SQL Injection and Cross-Site Scripting (XSS) Defense:
Server side defense using Prepared Statement is the most effective way to protect from SQL Injection, because it ensures that intent of query is not changed. It is very important to lockdown database server and to follow the Principle of Least Privilege . Modern web applications also rely heavily on caching and database schema design to improve performance . For prevention code injection attacks, including SQL Injection and XSS all user data should be validated. Input validation can be performed client side using JavaScript, but from security prospective it is not effective, because it doesn't provide protection for server-side code. Despite rule that input must be validated server-side sometimes validation should be performed client-side . Web frameworks and filters that offer automate sanitization to prevent XSS in web applications are gaining popularity, because manual implementation of input sanitization in web application is prone to errors.

4.3.Secure Session and Token generation for preventing CSRF attack

Preventing CSRF requires the inclusion of a unpredictable token as part of eachtransaction. Such tokens should at a minimum be unique per user session, but can alsobe unique per request.
1. The preferred option is to include the unique token in a hidden field. This causes thevalue to be sent in the body of the HTTP request, avoiding its inclusion in the URL,which is subject to exposure.
2.The unique token can also be included in the URL itself, or a URL parameter.However, such placement runs the risk that the URL will be exposed to an attacker, thus compromising the secret token. so token must be kept hidden.
The Filtering module
The Filtering module is first registered in the Java server based web application, then it intercepts each incoming http request and passes that request to the restore view phase of JS life cycle. It consists of a new tag library which is responsible for the adding new random tokens in the JS page upon each new http request from the user.
• Adds a new random token for each form during each http response;
• Validates the form token with the token stored in the session for that user in each http request, if the token is changed or missing, the application will generate the appropriate exception.
This module provides protection against Cross-site request forgery (CSRF), since another page would not know the value of this token and csrfguard from the OWASP does not offer integration with JSF based web application.

4.4. One time session cookies and use SSL for encrypted communication.

Web applications are built on the stateless HTTP protocol, so session management is an application-level responsibility. Session security is critical to the overall security of an application.
The following practices improve the security of your Web application's session management:
Use SSL to protect session authentication cookies.
Encrypt the contents of the authentication cookies.
Limit session lifetime.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 11, November 2014

Do not pass authentication cookies over HTTP connections. Set the secure cookie property within authentication cookies, which instructs browsers to send cookies back to the server
only over HTTPS connections. Reduce the lifetime of sessions to mitigate the risk of session hijacking and replay attacks. The shorter the session, the less time an attacker has to capture a session cookie and use it to access your application.

Fig.4.4:Secure Session Formation

4.5. Validation of Private Object Reference

The best way to protect the application against direct object reference attacks is through the validation of private object references. Others include the avoidance of the exposition of private object references to the users. For example primary keys or filenames use the index, indirect reference map, or other indirect methods that can be easily validate. If the user uses
the direct object, then it first ensures that the user is authorized and then exposed URL with indexing parameter such as "http://helloworld.com/file.jsp?file=1"
sets the "file" parameter to "1" value. If the application exposes direct references to the database structures, then it ensures that Sql statements and other database access methods only allow authorized records.

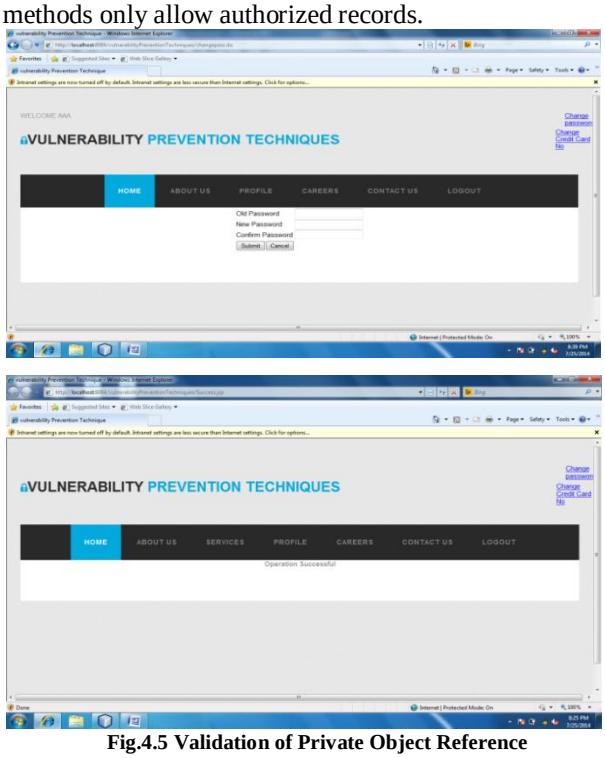

Fig.4.5 Validation of Private Object Reference

4.6. Encrypt password using SHA-256 algorithm

Too often passwords are stored as clear text. Thus the password can be read directly by the database's administrator, super users or SQL Injection attack etc. The backup media is also vulnerable. In order to solve this problem, passwords must be stored encrypted. For this we are using One way functions (SHA-256) also known as Hashing functions Passwords are secrets. There is no reason to decrypt them under any circumstances. Admin should be able to set new passwords not read back old passwords. Therefore, there is no reason to store passwords in a reversible form.

4.7. Creating Security Matrix to Restrict URL

The security experts or the developers need to plan authorization by creating a security matrix that maps the roles to the functions of the application. It is a key step to provision of protection against unrestricted URL access. The Web application not only provides access control to the URL, but also
confirms to the business logic residing in the application. As it generally happened that the access control is placed into the presentation layer, but it leaves the business layer unprotected. Moreover, It is also not sufficient to ensure only once during the process that the user is authorized to resources and then leaves it unchecked during the subsequent steps. Otherwise, the attacker may skip the steps of authorization and forge the parameter value necessary to continue on the next steps. One should assume that the users might be aware of the special or hidden URLs or API and provide protection against

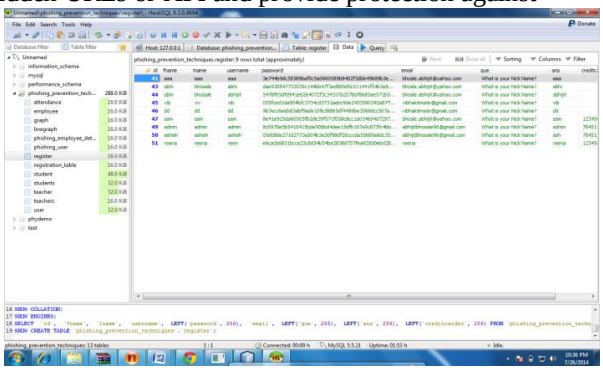

Fig.4.6: Encrypt password using SHA-256 algorithm

5.CONCLUSION

In present era of online world internet is the home of hackers and vulnerable web applications are their bank. So web applications must provide security and assurance. Thats why secure coding practices are use to create such applications with high security. It has been seen that after applying all secure coding practices and mitigation techniques to be secure from attacks, that are being used by hacker to steal information are prevented. It has been proved that Apart from security web applications must be developed using SDLC for early vulnerability detection and prevention.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 11, November 2014

REFERENCES

[1] Microsoft SDL(Security Development Lifecycle) and Building Secure Applications. Official SDL Web Site: http://www.microsoft.com/sdl

[2]. Atefeh Tajpour CASE & Mohammad Zaman Heydari: SQL Injection Detection and Prevention Tools Assessment, IEEE, 2010.

[3]. William G.J. Halfond and Alessandro Orso: Preventing SQL Injection Attacks Using AMNESIA

[4]. Lwin Khin Shar and Hee Beng Kuan Tan:Defending against Cross-Site Scripting Attacks,IEEE,2012

[5]. Tatiana Alexenko Mark Jenne & Suman Deb Roy Wenjun Zeng, Columbia: Cross-Site Request Forgery: Attack and Defense, IEEE,2010

[6]. Sreenivasa Rao & Kumar N: Web Application Vulnerabilities Assessment and Preventing Techniques, Int. J. of Enterprise computing and business System., Vol. 2, No. 1, January,2012

[7]. Daniel Huluka DSV and Oliver Popov DSV: Root Cause Analysis of Session Management and Broken Authentication Vulnerabilities, IEEE, 2012.

[8]. Goodman, G.; West, G., Jr.; Schoenfeld, I.; , "Criteria for review of root-cause analysis programs," Human Factors and Power Plants, 1997. 'Global Perspectives of Human Factors in Power Generation'. Proceedings of the 1997 IEEE Sixth Conference on, vol., no., pp.2/1-2/6, 8-13 Jun 1997

[9]. Cross Site Scripting-Latest developments and solutions-A survey: JayamsakthiShanmugam1, Dr. M. Ponnavaikko2,: Int. J. Open Problems Compt. Math., Vol. 1, No. 2, September 2008

[10]. Ziqing Mao, Ninghui Li, Ian Molloy: Defeating Cross-Site Request Forgery Attacks with Browser-Enforced Authenticity Protection

[11]. WilliamZeller and Edward W. Felten:Cross-Site Request Forgeries: Exploitation and Prevention

[12]. M. Johns and J. Winter. RequestRodeo: Client side protetion against session riding. In Proceedings of the OWASP Europe 2006 Conference, 2006

[13]. Aanchal Jain & Vineet Richariya: Implementing a Web Browser with Phishing Detection Techniques ,World of Computer Science and Information Technology Journal (WCSIT) , Vol. 1, No. 7, 289-291, 2011

[14] Rakeshkumar Kachhadiya," Development of the Security Framework based on OWASP ESAPI for JSF2.0"Master Thesis - Applied Computer Science Albert-Ludwigs-Universität Freiburg im Breisgau, 2 May 2012

[15] Curphey, M., Wiesman, A., Van der Stock, A., Stirbei, R.: "A Guide to Building Secure Web Applicationsand Web Services". OWASP (2010).

[16] Halfond WGJ, Viegas J, Orso A. A classification of SQL-injection attacks and countermeasures.Proceedings ofthe International Symposium on Secure Software Engineering, Washington, DC, U.S.A., March 2006.