Building A Secure & Anti-Theft Web Application By Detecting And Preventing Owasp Critical Attacks- A Review

Vibhakti Mate[1], Milind Tote[2], Abdulla Shaik[3]

1,2,3, Department of Computer science & Engineering, Nuva college of Engg. & Tech., Nagpur (MH),India

vibhaktimate@gmail.com[1],milind_tote@gmail.com[2], snanmka@gmail.com[3]

ABSTRACT-

Web applications of all kinds, whether online shops or partner portals, have in recent years increasingly become the target of hacker attacks. The attackers are using methods which are specifically aimed at exploiting potential weak spots in the web application software itself - and this is why they are not detected, or are not detected with sufficient accuracy, by traditional IT security systems. OWASP develops tools and best practices to support developers in the development and operation of secure web applications. Additional protection against attacks, in particular for already productive web applications, is offered by what is still emerging category of IT security systems, known as Web Application Firewalls often also called Web Application Shields or Web Application Security Filters. 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. In this paper we are implementing four vulnerabilities of web application i.e., SQLI, CSRF, XSS and Broken Authentication and session management to find out their prevention strategies over existing web application.

The main objective of this paper 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 above 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 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

Web applications are widely used to provide functionality that allows companies to build and maintain relationships with their customers. The information stored by web applications is often confidential and, if obtained by malicious attackers, its exposure could result in substantial losses for both consumers and companies. Recognizing the rising cost of successful attacks, software engineers have worked to improve their processes to minimize the introduction of vulnerabilities. In spite of these improvements, vulnerabilities continue to occur because of the complexity of the web applications and their deployment configurations. The continued prevalence of vulnerabilities has increased the importance of techniques that can identify vulnerabilities in deployed web applications.

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 &show how these attacks are used to compromise user identity and credentials.

For providing better security and assurance of confidentiality web applications must be developed with secure coding practices. So for securing web application from any attack from web we will create a web application with vulnerability prevention and we will test this application with different attacks that can be perform on it

Designing secure authentication and session management mechanisms are just a couple of the issues facing Web application designers and developers. Other challenges occur because input and output data passes over public networks. Preventing parameter manipulation and the disclosure of sensitive data are other top issues.

Some of the top issues that must be addressed with secure design practices are shown in Figure

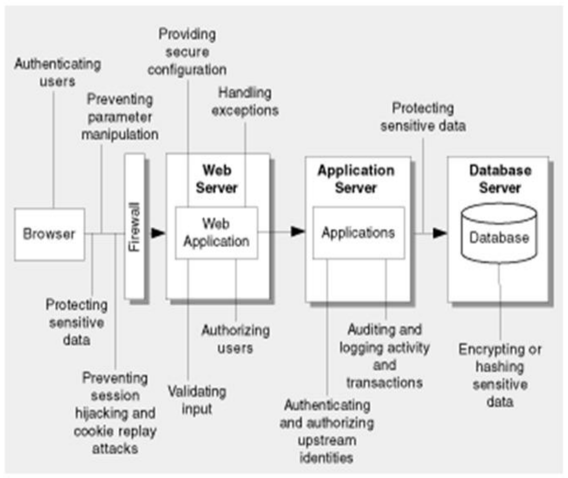
Authenticating users
Providing secure configuration
Preventing parameter manipulation
Handling exceptions
Protecting sensitive data
Web Server
Application Server
Database Server
Browser
Firewall
Web Application
Applications
Database
Protecting sensitive data
Authorizing users
Auditing and logging activity and transactions
Encrypting or hashing sensitive data
Preventing session hijacking and cookie replay attacks
Validating input
Authenticating and authorizing upstream identities

Figure 1.0: Solution Architecture

This paper proposes a novel specification-based methodology for the prevention of above(abstract) listed Attacks. The two most important advantages of the new approach against existing analogous mechanisms are that, first, it prevents all forms of above listed attacks; second, Current technique does not allow the user to accessdatabase directly from the database server. Our proposed framework for building secure and anti-theft web applications is consisting of four stages. In each stage we analyze the inputted data taken from the user and make a decision, whether that is suspected or not.

2. ATTACKS ON WEB APPLICATIONS

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.

2.1 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. 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

3.0 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 four different attacks on web applications, to show how we can prevent from these to achieve secure web application.

3.1. SQL-Injection(SQLI):

An SQL injection attack (SQLIA) 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.

3.2 SQL Injection Types:

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

3.2.1. Tautology:

This attack bypasses the authentication and access data through vulnerable input field using "where" clause by injecting SQL tokens into conditional query statements which always evaluates to true.

3.2.2. Logically incorrect queries:

The error message sent from database on being sending wrong SQL query may contain some useful debugging information. This could help in finding parameters which are vulnerable in the web application and hence in the database of the application.

3.2.3. Union queries:

The “Union” keyword in SQL can be used to get information about other tables in the database. And if used properly this can be exploited by attacker to get valuable data about a user from the database

3.2.4. Piggy-backed Queries:

This is the kind of attack where an attacker appends “;” and a query which can be executed on the database. It could be one of the very dangerous attacks on database which could damage or may completely destroy a table. If this attack is successful then there could be huge loss of data.

3.2.5. Stored Procedure:

It is an abstraction layer on top of database and depending on the kind of stored procedure there are different ways to attack. The vulnerability here is same as in web applications. Moreover all the types of SQL injection applicable for a web application are also going to work here.

3.2.6. Blind Injection:

It’s difficult for an attacker to get information about a database when developers hide the error message coming from the database and send a user to a generic error displaying page. It’s at this point when an attacker can send a set of true/false questions to steal data.

3.2.7. Timing Attacks:

In this kind of attack timing delays are observed in response from a database which helps to gather information from a database. SQL engine is caused to execute a long running query or a time delay statement with the help of if-then statement which depends on the logic that has been injected. It is possible to determine whether injected statement was true or false depending on how much time condition is true this code is injected to produce response delay in time.

3.3 Existing system

Many existing techniques rely on complex static analyses in order to find potential vulnerabilities in the code. Some techniques also based on dynamic negative tainting [1] that focuses on untrusted data. These kinds of conservative static analyses can generate high rates of false positives and can have scalability issues when applied to large complex applications. These techniques involve extensive human effort. They require developers to manually rewrite parts of the Web applications, build queries using special libraries, or mark all points in the code at which malicious input could be introduced.

3.4 Proposed System

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.

4.1Cross-Site scripting (XSS):

Cross-site scripting (XSS) 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. Un- fortunately, 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 solely depending on the security of the web application.

4.1.1 Existing System

Existing techniques for defending against XSS exploits suffer from various weaknesses: inherent limitations, incomplete implementations, complex frameworks, runtime overhead, and intensive manual-work requirements. Security researchers can address these weaknesses from the below perspectives. From a development perspective, researchers need to craft simpler, better, and more flexible security defenses. They need to look beyond current techniques by incorporating more effective input validation and sanitization features. In time, development tools will incorporate security frameworks that implement state-of-the-art technology[8].

4.1.2 Proposed System:

Cross-site scripting (XSS) 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 abouttobetransferredtoathirdparty(i.e., theadversary), 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 controldependencies,weadditionallyemployanauxiliary,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, we automatically tested the enhanced browser on more than one million web pages by means of a crawler that is capable of interpreting JavaScript code. The results of this large-scale evaluation demonstrate that only a small number of false positives is generated, and that our underlying concepts are feasible in practice.

4.2 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. 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.

4.2.1 Existing System

Several defense mechanisms have been proposed for CSFR attacks, we now discuss their limitations [11]. CSRF attacks are particularly difficult to defend because cross-site requests are a feature of the web application. To effectively defend against CSRF attacks, one needs as much information about an HTTP request as possible, in particular, how the request is triggered and crafted. Such information is available only within the browser. Existing defenses suffer from the fact that they do not have enough information about HTTP requests. They either have to change the web application to enhance the information they have or to use unreliable source of information (such as Referer header)[10]. Even when such information is available, it is still insufficient. The most popular CSRF defense is to authenticate the web form from which an HTTP request is generated. CSRF attacks use HTTP requests that have lasting observable effects at the web site. Two request methods are used in real-world HTTP requests: GET and POST [9]. The GET method, which is known as a "safe" method, is used to retrieve objects. The GET requests should not have any lasting observable effect (e.g., modification of a database). The operations that have lasting observable effects should be requested using the method POST. The POST requests have a request body and are typically used to submit forms. However, there exist web applications that do not follow the standard and use GET for requests that have lasting side effects. Web pages in one site may result in HTTP requests to another site; these are called cross-site requests.

4.2.2 Proposed system

The fundamental nature of the CSRF attack is that the user's browser is easily tricked into sending a sensitive request that does not reflect the user's intention. Our solution to this problem is to directly address the confused deputy problem of the browser. More specifically, we propose Browser-Enforced Authenticity Protection (BEAP), which enhances web browsers with a mechanism ensuring that all sensitive requests sent by the browser reflect the user's intention. BEAP achieves this through the following. First, BEAP infers whether an HTTP request reflects the intention of the user. Second, BEAP infers whether authentication tokens associated with the HTTP request are sensitive. An authentication token is sensitive if attaching the token to the HTTP request could have sensitive consequences. Third, if BEAP concludes that an HTTP request reflects the user's intention, the request is allowed to be sent with authentication tokens attached. If BEAP concludes that an HTTP request may not reflects the user's intention, it strips all sensitive authentication tokens from the HTTP request [4].

4.3 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.

4.3.1 Existing system

For the attack like session management and broken authentication, possible employment of Root Cause Analysis (RCA) has been developed. It provide some security aspects of web applications.RCA has a proven record as an effective problem analysis method in different knowledge domains that may range from handling customer complaints on a limited products line to critical infrastructures such as risk analysis in nuclear power plants [7]. We feel that the inherent potential of RCA is somehow ignored relative to computer and information security. Usually, it has been reduced to certain applications in the analysis of very few specific attacks on computer networks and data centers. Additional work is required to identify causes of problems in the web applications such as the generic and high level research presented in.

4.3.2 Proposed system:

Our research focuses on the application of RCA to identify root causes of Session Management and Broken Authentication Vulnerabilities and eventually come up with solutions that shall minimize the recurrence of these vulnerabilities in web applications. In Our paper we are selected Reality Charting Root Cause [6] analysis software as an approved tool for the implementation of the RCA methodology which is one of the commonly used RCA methods in different domains. It is a simple causal process tool whereby one (1) asks why the (defined) problem occurs, (2) answers the question with at least two causes then (3) examines each previous cause until there are no more causes attributable to the last identified causes.

In this section for Session management ,to strengthen 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 (for instance by using hashing or encryption). These solutions protect authentication data but the solution creates communication overhead and need further optimization.

5.IMPLEMENATATION

Our Prototype currently includes separate proposed techniques for each type of above mentioned attacks .In practical we are implementing four attacks i.e., SQLI,XSS,CSRF,BA&SM, to suggest the prevention over them to achieve highly secure web application.

CONCLUSION

In this paper we describe Vulnerability Assessment (VA) process of identifying, quantifying, and prioritizing the vulnerabilities (security holes) in a system. Despite numerous attempts to counter attacks, vulnerabilities are frequently found in most web applications. Therefore, our above [Enhancement] proposed techniques reduce the rate of strategies of attacks in most of the web applications. Depending upon the response given by our proposed techniques we are able to qualified and identified problems on most web applications.

REFERENCES

1. Atefeh Tajpour CASE & Mohammad Zaman Heydari: SQL Injection Detection and Prevention Tools Assessment ,IEEE,2010.
2. William G.J. Halfond and Alessandro Orso: Preventing SQL Injection Attacks Using AMNESIA
3. Lwin Khin Shar and Hee Beng Kuan Tan:Defending against Cross-Site Scripting Attacks,IEEE,2012
4. Tatiana Alexenko Mark Jenne & Suman Deb Roy Wenjun Zeng, Columbia: Cross-Site Request Forgery: Attack and Defense ,IEEE,2010
5. 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
6. Daniel Huluka DSV and Oliver Popov DSV: Root Cause Analysis of Session Management and Broken Authentication Vulnerabilities, IEEE, 2012.
7. 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
8. Cross Site Scripting-Latest developments and solutions-A survey :Jayamsakthi Shanmugam1, Dr. M. Ponnavaikko2,: Int. J. Open Problems Compt. Math., Vol. 1, No. 2, September 2008
9. Ziqing Mao, Ninghui Li, Ian Molloy:Defeating Cross-Site Request Forgery Attacks with Browser-Enforced Authenticity Protection

10. William Zeller and Edward W. Felten:Cross-Site Request Forgeries: Exploitation and Prevention
11. M. Johns and J. Winter. RequestRodeo: Client side protetion against session riding. In Proceedings of the OWASP Europe 2006 Conference, 2006
12. 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