ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 9, September 2015

Securing Web Applications from SQL Injection using OAuth Framework

Darshini C N[1], Dr. Mydhili K Nair[2]

Student, M. Tech (Software Engg), Dept. of Information Science Engg, M S Ramaiah Institute of Technology, Karnataka, India[1]

Associate Professor, Dept. of Information Science Engg, M S Ramaiah Institute of Technology, Karnataka, India[2]

ABSTRACT: In recent years database-driven applications are largely deployed on the internet and many organizations need to store sensitive information or private documents and use them for providing services to their customers. These applications can contain confidential information which are highly valuable and thus an ideal target for attacks. Most of the web application attacks are through cross-site scripting (XSS) and SQL injection.SQL injection has become a serious threat to web applications that reads user inputs and uses it to make crafted SQL queries for an underlying database; as a result an illegal query will be provided to the database. SQLIAs are caused by insufficient validation to the inputs provided by the users. In this paper, we have proposed OAuth authorization protocol framework which enables an application to obtain limited access by introducing an authorization layer between the resource and third-party clients. The client has to make access requests to the resources that are in control of the resource owner and then a different set of credentials are issued that is an access token by the authorization server which is a string denoting the lifetime, scope and access attributes. The client should use this access token in order to access the resources hosted by the server. Therebyproviding a more robust security framework to the web application through OAuth.

KEYWORDS: SQL injection attack, OAuth framework protocol, authorization server, request token, access token, client.

I. INTRODUCTION

Web applications have become an integral part of our daily routine. Information sharing through web and business adoption on web has been increased due to emergence of web 2.0 which is used as a base for delivering the services and hence websites become targets for attacks.Security in the web has become important while making any transaction. The developers need to build applications in such a way that they would be secure enough when being executed in a partially trusted or untrusted environment since most of the businesses have adopted websites for delivering the content or services and interacting with their customers.

Authentication and authorization has been given more importance while considering security of web applications since it is necessary for knowing not only what the user is doing and who the user is but also if he is allowed to do that. Authorization and authentication is a term for intelligently controlling access to computer resources, auditing usage, and also providing the information essential for services. The combination of these processes is considered important for effectively managing the security.In large distributed networked applications, security administration is complex and prone to errors since the access control lists are specified for each user individually. Role based authorization is a technique which is gaining popularity because it reduces the complexity and also the cost of security. In role based authorization the authenticated users are assigned specific roles and these roles are granted privileges.

The major web application attacks are through SQL injections and Cross-site Scripting (XSS) attacks. These attacks are caused because of improper validation of the user inputs, flaws in the code. As a result, the victims may have financial damage as their identity is stolen or the user's resources could be stolen or damaged. Web applications make

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 9, September 2015

use of the input from HTTP request in order to determine the responses. The attackers are able to tamper with HTTP request including query string cookies, URL and so on.

SQL Injection is a code insertion technique that is used to achieve access to the database of a web application. This vulnerability occurs when unfiltered user input are embedded to SQL statements and then executed. It allows an attacker to retrieve information that is crucial from databases on the web server. In recent years database-driven applications are largely deployed on the internet and many organizations need to store sensitive information or private documents and use them for providing services to their customers. These applications can contain confidential information which are highly valuable and thus an ideal target for attacks.

SQL injection attacks (SQLIA), a class of code injection attacks is one of the most known vulnerability that targets data-driven applications. Using SQL injections an attacker can spoof identity, retrieve sensitive data from the database, tamper with existing data in the database, disclosure of all the data on the system, destroy the data or make it unavailable and in some cases become administrators of the database. These attacks have become a serious threat to web applications that reads user inputs and uses it to make SQL queries for an underlying database. In this type of attack specially crafted input strings are used, as a result an illegal query will be provided to the database.

SQLIAs are caused by insufficient validation to the inputs provided by the users and through this an attacker can provide inputs with embedded SQL commands directly to the database. Although the vulnerabilities that lead to SQLIAs are well understood it continues to be a problem because of lack of effective techniques for detecting and preventing them and there are few analysis based techniques that target SQLIAs.

A typical web application structure is shown in figure. 1 and applications with such architecture constructs database queries dynamically using the input provided by the user and send them to the database through APIs for execution.Security problem arises when the inputs are not handled properly. SQLIAs occur when attackers pass crafted string inputs as part of the query that causes the application to generate and send a query which is an unspecified behavior of the application. As a result confidential data such as personal information and financial records will be lost.

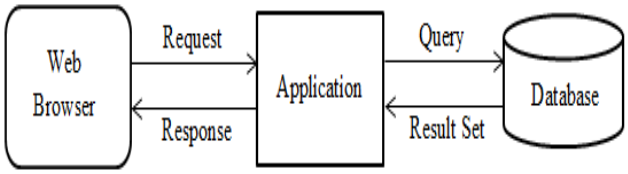

Fig. 1: Web application structure

The use of web applications for online transactions and others has been increased and hence it should be secure and reliable.In spite of understanding the vulnerabilities of SQLIAs it continues to be a problem because of lack of efficiency in addressing the problem. The attackers continue to find new methods for exploiting the input validation used by programmers. The testing techniques designed for SQL injections provide partial solutions to the problem since the developers often neglect testing process.

II. RELATED WORK

We discuss few of the work that has been done to provide new techniques for prevention and detection of SQL injection attacks. SQL injections occur when the data given by the user is directly embedded in a SQL query and is not validated. The attackers submit input strings containing database commands encoded in it [2]. Different kinds of SQLIAs known are discussed in [3, 4] that includes SQL tautologies, union based query, piggy-backed queries and so on.

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 9, September 2015

In traditional authentication models, the client has to request for a restricted resource by authenticating with server. For providing access to third-party applications the resource owner's credentials had to be shared with the third-party with resource owner's not able to restrict their access to limited set of resources.

From reference [2], the authors have presented a tool called WASP which implements the idea of positive tainting and concept of syntax-aware evaluation. Unlike dynamic tainting, the positive tainting explicitly identifies the trusted data in the program which eliminates the false positives resulting from incomplete identification. Even though the coding practices have proved to prevent SQLIAs theoretically, they are less effective in addressing the problem.

The pre-deployment techniques fall under the static analysis category [5]. A static analysis framework [7] that operates on the source code to prevent tautology attacks uses static analysis to obtain a set of SQL queries which would be generated as finite state automata. An algorithm generates the automaton to check if there is a tautology that could be a threat. The limitation here is only tautology based SQLIAs are detected. A JDBC checker [8] detects SQL injections caused by incorrect type checking of inputs given by users. General form of SQLIAs is not detected since they are syntactically correct queries.

The post-deployment techniques include dynamic analysis technique to detect SQLIAs. A runtime technique to eliminate SQL injection [9] compares the parse tree of SQL statement before inclusion and after inclusion of the user input. It uses secret key to delimit the user input during parsing and thus security depends on attacker not being able to discover the key. AMNESIA [10] for detection and prevention of SQLIAs combines static and dynamic analysis. In static phase, query models are built and in dynamic phase, the dynamically generated queries are checked against the model built in static phase. The accuracy of this technique depends on the static analysis of query model building.

III. METHODOLOGY AND STRATEGY

The system makes use of the OAuth authorization protocol which enables any third-party application to obtain limited access by introducing an authorization layer between the resource and clients. The client has to make access requests to the resources that are in control of the resource owner and then a different set of credentials are issued. Rather than providing owner's credentials the client is given an access token by the authorization server which is a string denoting the lifetime, scope and access attributes. The client should use this access token in order to access the resources hosted by the server.

OAuth addresses the issues specified with traditional authentication model by introducing authorization layer. In order to access the resources the client has to take permission expressed in the form of a token from the resource owner. The use of token makes sure that the resource owner will not share his credentials with the client unnecessarily. It also helps in specifying the scope, lifetime as well as it can be revoked.

OAuth, an open standard for authorization provides client applications a secure access to resources without sharing the owner's credentials, instead it allows access tokens to the third-party clients as shown in figure 2. The client application which requests to resources has to be registered first with the authorization server that is associated with the resource server. A client ID and a client secret is assigned to the client application when it is registered. If client application is registered with many authorization servers then each of the servers issues their own unique client ID and client secret. OAuth uses HTTPS for communication between client and authorization server.

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 9, September 2015

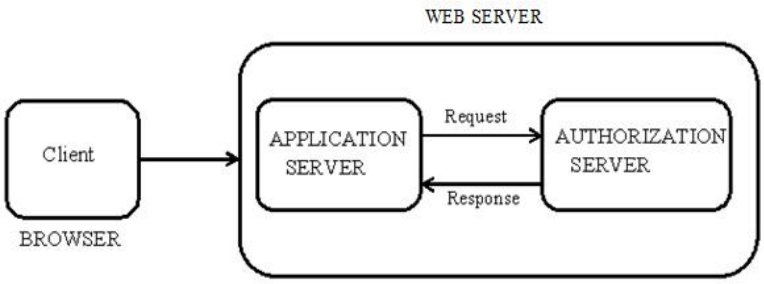

Fig. 2: Architecture of OAuth protocol

The OAuth protocol has defined four roles.

1. Resource owner which is an entity that provides access to a protected resource to the end-user. It can be a person or application who owns data that would be shared.
2. Resource server which hosts the resources that are protected and capable of providing response to protected resource requests with the access tokens.
3. Authorization server is a server responsible for providing access tokens after successful authorization of resource owner to the clients.
4. Client which is a third party-application that attempts to access protected resource on behalf of resource owner with its authorization.

A. *Tokens*

The authorization server randomly generates strings called tokens which are issued along with the client requests. There are two types of token – access token and refresh token.

- Access tokens are most important since it provides access to the user data for a third-party application. The access token is passed as a parameter or as header in the request that will be sent to the resource server. The authorization server defines the lifetime which is usually limited for the access token.
- Refresh token is provided along with the access token but not sent to the resource server for each request from the client. When access token is expired the refresh token is used for renewing it.

B. *Client Types*

During the registration of the client with the authorization server, the client has to specify the client type, include information required by authorization and also provide the redirection URI. The client type is based on definition of secure authentication given by the authorization server and the exposure level of the credentials. There are two client types defined based on the ability of secure authentication.

1. Confidential. The clients are capable of preserving the confidentiality of their credentials.
2. Public. The clients are incapable of preserving the confidentiality of their credentials.

On registering with the authorization server, a unique string called client identifier which represents the registration information is provided. Based on the client type the authorization server establishes a suitable method for the security requirements. The authorization server supports the client credentials in request body using client_id and client_secret parameters. The client_id is the client identifier which is provided during registration process.

C. *Protocol Endpoints*

The authorization uses two server endpoints.

1. Authorization endpoint
2. Token endpoint

Authorization endpoint used for obtaining authorization from resource owner using user's redirection. It acts as the interface for obtaining an authorization grant from the resource owner. The authorization server has to verify the identity of resource owner. The requests made to the authorization endpoint results in user authentication with the

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 9, September 2015

transmission of clear-text credentials. The HTTP GET method has to be used while making interaction with the authorization endpoint.

Token endpoint used for exchanging authorization grant with access token. The client can also use this endpoint for exchanging the refresh tokens with the access tokens. This endpoint is used with every authorization grant. The results of the request to token endpoint also provide transmission of text credentials as with authorization endpoint. The client must use the HTTP POST method while accessing the access token request with token endpoint.

The protocol also makes use of client endpoint which is a redirection endpoint used for returning authorization credential response to the client from the authorization server. When the interaction with resource owner is completed, the user-agent is redirected back to the client by the authorization server. The redirection endpoint which was previously established during the client registration or authorization request is used by authorization server for redirecting the user back to the client application.

The authorization server notifies the resource owner or automatically redirects the user-agent to invalid redirection URI if there is any failure in the authorization request because of missing redirection URI or any invalid redirection URI.

IV. IMPLEMENTATION

The OAuth framework has been developed in JAVA and MySQL server has been used as the database.The authorization server interaction has been implemented in the OAuth provider and all the client-side interactions in the OAuth example modules.

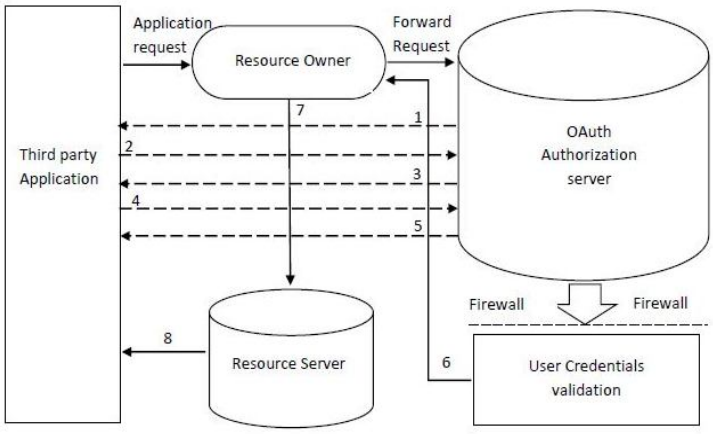

Fig. 3: Protocol flow

The figure 3 shows the flow in OAuth process where a client requests for authorization access from the resource owner. The same has been explained in the algorithm.

A. *Algorithm*

Step 1: Register the application with OAuth server.

Step 2: OAuth server provides the authorization page for the user. User enters his/her credentials.

Step 3: OAuth server provides a request token if the user credentials are correct.

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 9, September 2015

Step 4: The request token is exchanged for an access token.

Step 5: If the request token is alive and true, an access token is given to the application.

Step 6: If the access token is correct, access to resource is granted from OAuth Server and the details are given to the resource owner.

Step 7: Based on the response from the OAuth Server, resource owner provides access to the resource server.

Step 8: The content from the resource server is then passed to the requesting third party application.

V. RESULTS

The developed solution has been employed in an application for evaluation purpose. If an illegitimate query is given to the database the query is not allowed to pass through since the mechanism of exchange of tokens has been used. This makes it secure as the tokens provided are validated before giving any access to the protected resource.

Threat	Secure	Reason
Accessing client secret	Yes	The web configuration is not accessible from the client application.
Accessing refresh tokens	Yes	Since the client id is validated in the OAuth provider.
Accessing access tokens	Yes	The tokens are encrypted and secured in database.
Attackers obtain authorization	Yes	The provider authorizes the clients before returning the authorization codes and also validates the redirect UI
Leaking or eavesdropping authorization code	Yes	Since nonce table is used which ensures one time use

Table 1:Proof of OAuth against threats

The servers are also provided with minimum privileges to the database and the credentials are normally hashed or encrypted rather than in clear text. The tokens are generated for one time use and are not stored. The OAuth framework itself is secure as shown in table 1. In the table, the client secret cannot be accessed since the web configuration is not accessible from client application and other threats and the reasons for their secure are explained in the table.

VI. CONCLUSION

In this paper, we have discussed about the existing approaches for preventing SQL injections along with their advantages and disadvantages. We have introduced OAuth protocol for eliminating the web application's security attack like SQL injection attack. This protocol helps in making authorization decisions over the network and also provides limited access to any HTTP service in a simple method. The key advantage is that the resource owners can

ISSN(Online): 2319-8753
ISSN (Print) : 2347-6710

International Journal of Innovative Research in Science, Engineering and Technology

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 9, September 2015

share their resources without providing any important confidential information since they are handled through access tokens, specifies what the client or third-party applications can access. Thereby providing a more robust and security framework to the web application.

REFERENCES

[1] RamyaDharam and Sajjan G. Shiva, Runtime Monitoring Technique to handle Tautology based SQL Injection Attacks, International Journal of Cyber-Security and Digital Forensics (IJCSDF) 1(3): 189-203,2012
[2] W. G. J. Halfond, A. Orso and P. Manolios, "Using Positive Tainting and Syntax-aware Evaluation to Counter SQL Injection Attacks", Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2006.
[3] W. G. J. Halfond, J. Viegas, and A.Orso, "A Classification of SQL - Injection Attacks and Countermeasures", Proceedings of the IEEE International Symposium on Secure Software Engineering, 2006.
[4] A. Tajpour and M. Massrum, "Comparison of SQL Injection Detection and Prevention Techniques", In 2nd International Conference on Education Technology and Computer, 2010.
[5] W. G. J. Halfond and A. Orso, "Combining Static Analysis and Runtime Monitoring to Counter SQL Injection Attacks", Proceedings of 3rd International Workshop on Dynamic Analysis, 2005.
[6] Software Security Testing, Software Assurance Pocket Guide Series: Development, Volume III, Version 1.0, May 21, 2012.
[7] G. Wassermann and Z. Su, "An Analysis Framework for Security in Web Applications", Proceedings of the FSE Workshop on Specification and Verification of Component Based Systems, 2004.
[8] C. Gould, Z. Su and P. Devanbu, "JDBC Checker: A Static Analysis Tool for SQL/JDBC Applications", Proceedings of the 26th International Conference on Software Engineering, 2004.
[9] G. T. Buehrer, B. W. Weide and P. A. G. Sivilotti, "Using Parse Tree Validation to Prevent SQL Injection Attacks", International Workshop on Software Engineering and Middleware, 2005.
[10] W. G. Halfond and A. Orso, "AMNESIA: Analysis and Monitoring for Neutralizing SQL-Injection Attacks", Proceedings of the IEEE and ACM International Conference on Automated Software Engineering, Nov 2005.
[11] draft-ietf-oauth-v2-23 - The OAuth 2.0 Authorization Framework, http://tools.ietf.org/html/draft-ietf-oauth-v2-23
[12] OAuth 2.0 Tutorial _ tutorials.jenkov.com, http://tutorials.jenkov.com/oauth2/index.html
[13] Protocol Workflow _ hueniverse, http://hueniverse.com/oauth/guide/workflow/
[14] OAuth Authentication, http://nouncer.com/oauth/authentication.html
[15] Introduction to Web Application Security, http://enterprisewebbook.com/ch9_security.html
[16] Securing RESTful Web Services Using Spring and OAuth 2.0, HUGHES SYSTIQUE, http://www.hsc.com/Portals/0/Uploads/Articles/WP_Securing_RESTful_WebServices_OAuth263
[17] RFC 6819 - OAuth 2.0 Threat Model and Security Considerations.html, http://tools.ietf.org/html/rfc6819
[18] RFC 6750 - The OAuth 2.0 Authorization Framework_ Bearer Token Usage.html, http://tools.ietf.org/html/rfc6750
[19] End User Authentication with OAuth 2.0 — OAuth.html, http://oauth.net/articles/authentication/
[20] Request and Response _ OAuth2 Server PHP.html, https://bshaffer.github.io/oauth2-server-php-docs/overview/response/
[21] Hosting an OAuth2.0 Service in Passport.js with Mongoose » z43 Studio.html, http://z43studio.com/2014/07/oauth2
[22] Configuring+OAuth.htm, https://confluence.atlassian.com/display/GADGETS010/