ISSN: 2320-1363

Secured Data Communication in Cloud Computing using Channel API with MD5 Hashing

E. Srimathi*1, S. Geetha*2, I. Anette Regina*3

MPhil, Research Scholar, Muthurangam Govt. Arts College, Vellore, Tamilnadu, India

Assistant Professor, Department of CS, Muthurangam Govt. Arts College, Vellore, Tamilnadu, India

Associate Professor, Department of CS, Muthurangam Govt. Arts College, Vellore, Tamilnadu, India

Abstract

Cloud computing offers the vision of a virtually infinite pool of computing, storage and networking resources where application can be scalable and deployed. The security threats on cloud increases rapidly. The threat starts from login module to the core storage. In this project different levels of threat is handled efficiently, which enables the secured data communications between the server and client. The login is authenticated using Open Authentication Protocol (OAuth) 2.0 with enhanced security mechanism. The data theft and other masquerading attacks are prevented using channel API (Application based Program Interface) which sends the data in a secured channel establish among the sender and the receiver using MD5(Message-Digest) Hashing. This is demonstrated by building an cloud based chat application which transfers the data from server to client and vice versa. It includes voice, non-voice and video chat communication. The Video Chat is demonstrated using WebRTC (Web Real-Time Communication).WebRTC provides page to page communication among HTML. Hence this project covers the prevention mechanism for several threats in cloud using different techniques.

I.INTRODUCTION

Cloud computing offers the vision of a virtually infinite pool of computing, storage and networking resources where application can be scalable and deployed. In particular Google cloud service provides Google App Engine for Java! With App Engine, you can build web applications using standard Java technologies and run them on Google's scalable infrastructure. The Java environment provides a Java 7 JVM, a Java Servlets interface, and support for standard interfaces to the App Engine scalable data store and services, such as JDO, JPA, Java Mail, and JCache. Standards support makes developing your application easy and familiar, and also makes porting your application to and from your own servlet environment straightforward. The Google Plug-in for Eclipse adds new project wizards and debug configurations to your Eclipse IDE for App Engine projects. App Engine for Java makes it especially easy to develop and deploy world-class web applications using Google Web Toolkit (GWT). The Eclipse plug-in comes bundled with the App Engine and GWT SDKs. Third-party plug-in is available for other Java IDEs as well.

The main uniqueness of this implies on its learning structure. We are enhancing the

ISSN: 2320-1363

features of this application by adding more functionality. One of the functionality we are adding is the calls through browsers where browser to browser communication is achieved for video chat. Text chatting based on individual and group is also available.

FLOW DIAGRAM

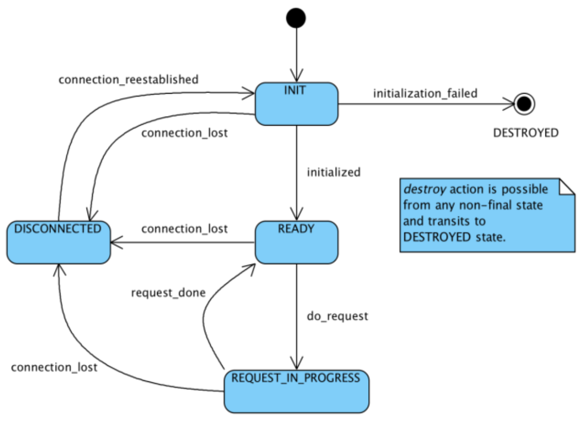

Fig 1: Data Flow Diagram

II.MODULE DESCRIPTION

1. Open Authentication

In general, this module deals with authenticating the user inside the application, this is considered to be the efficient and secured method in order to allow the user to be authenticated inside the application. An open authentication protocol is an open standard to authorization. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. Designed specifically to work with Hypertext Transfer Protocol (HTTP), OAuth essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner, or end-user. The client then uses the access token to access the protected resources hosted by the resource server.

Steps involved in OAuth:

(i). Obtain OAuth 2.0 credentials

Both Service Provider (SP) like Google and the application know OAuth 2.0 credentials such as a client ID and client secret. The set of values varies based on what type of application you are building. For example, a JavaScript application does not require a secret, but a web server application does.

(ii). Obtain an access token from the Google Authorization Server.

Before your application can access private data using a Google API, it must obtain an access token that grants access to that API. A single access token can grant varying degrees of access to multiple APIs. A variable parameter called scope controls the set of resources and operations that an access token permits. During the access-token request, your application sends one or more values in the scope parameter. There are several ways to make this request, and they vary by type of application you are building.

For example, a JavaScript application might request an access token using a browser redirect to Google, while an application installed on a device that has no browser uses web service requests. Some requests require an authentication step where the user logs in with their Google account. After logging in, the user is asked whether they are willing to grant the permissions that your application is requesting. This process is called user consent. If the user grants the permission, the Google Authorization Server sends your application an access token (or an authorization code that your application can use to obtain an access token). If the user does not grant the permission, the server returns an error.

(iii). Send the access token to an API.

After an application obtains an access token, it sends the token to a Google API in an HTTP authorization header. It is possible to send tokens as URI query-string parameters, but we don't recommend it, because URI parameters can end up in log files that are not completely secure. Also, it is good REST practice to avoid creating unnecessary URI parameter names.Access tokens are valid only for the set of operations and resources described in the scope of the token request.

(iv). Refresh the access token, if necessary.

Access tokens have limited lifetimes. If your application needs access to a Google API beyond the lifetime of a single access token, it can obtain a refresh token. A refresh token allows your application to obtain new access tokens. Final after getting the token from Google, the key is exchanged between the Google server and the web application, then if the results matches, then the user is allowed to enter in to the application.

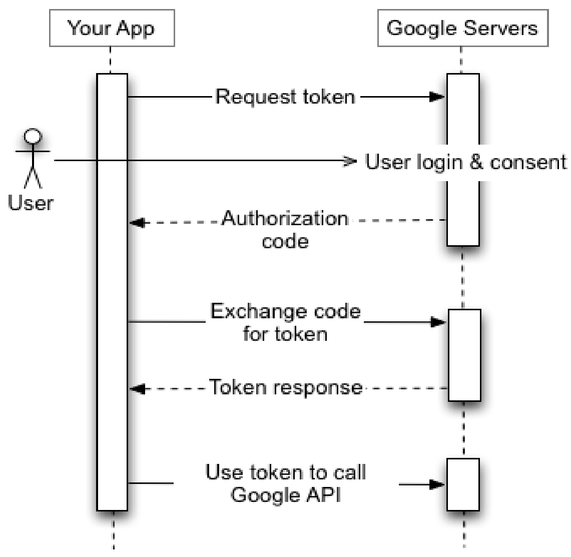

Fig 2: Open Authentication Process

2. Web Service Identification

In this module, the web services based on the functionalities required are generated. In general, each web services are intended to perform a separate operation. A web service repository is a set of disjoint services. We denote it as {w1, w2,….}

3. Functionality Request

As the growth of web related requirements increases gradually, the users who interact with the web needs the system to system interaction, created the need for a structure which provided interaction not only

ISSN: 2320-1363

with the user but also help application to application interaction. This problem called as Application Integration which is resolved by Web Services. Moreover, with the addition of the intelligence and autonomy of software agents, transactions may be equally automated for consumer-to-consumer, business-to-consumer, and business-to-business collaborations.

But it is a clear observation that number of web services is increased where a problem is raised to find an appropriate web service which satisfies user needs. The user wants an efficient way to find an appropriate web service which satisfied his needs in a short time. The functionality is mainly decided based on the input that is set for the web services and the output of the web services that the user needs. This is done in this module.

4. QoS Constraint

The objective of this module is to maximize an application-specific utility function under the end-to-end QoS constraints. Existing methods can be divided into two types: local selection method and global selection method. The local selection method is simple and efficient, but cannot meet the end-to-end QoS constraints. The global selection method can satisfy the global QoS constraints, but at the price of higher computational time, and it is not suitable for the dynamic environment.

To address this issue, an algorithm to combine global QoS constraints with local selection is proposed, which first splits the global QoS constraints into local constraints using heuristic method, and then uses local selection to find the optimal solution under the local constraints. It is intended to improve the performance by reducing the computation time greatly while achieving close-to-optimal results.

5. Operation Applicability

In this module, the web services are given an operation. After setting the operation the Web service is triggered. Then the implications evolved during the life time of the web service call are monitored. The variation is identified among the services where the operation differs one among the other as the time varies for each different web service calls.

5.1 Cross Domain Call

To make a cross domain call among web services, the Cross Origin Resource Sharing (CORS) is used which allows many resources from outside domain where the browser and the server are interact to determine whether or not to allow the cross-origin request.

Consider an example, a page from http://www.application1.com attempts to access a user's data in http://www.application2.com If the user's

ISSN: 2320-1363

browser implements CORS, then request header (i.e.) Origin: http://www.application1.com sends to application2. If an application2 allows the request, then it sends Access-Control-Allow-Origin: http://www.application1.com header in its response. If it does not allow the cross origin request, then browser sends an error to application1 instead of application2 response by using the following header Access-Control-Allow-Origin: *

This is applicable in the public content and it is intended to be accessible to everyone. It is related to JSONP technique for cross domain requests to make a cross domain call.

III. HTTP REDIRECT METHOD

The HttpRedirect concept is used to make a cross domain call to third party web services. The HttpRedirect property specifies the directory or URL to which a client is redirected when attempting to access a specific resource. For fast data retrieval this HttpRedirect is used.In the simplest configuration, it need only to set the enabled and destination attributes of the <httpRedirect> element in order to redirect clients to a new location.

III.1. Aim of HttpRedirect:

Firstly, transferring to another page using Redirect conserves server resources. Instead of telling the browser to redirect, it simply changes the "focus" on the Web server and transfers the request. This means you don't get quite as many HTTP requests coming through, which therefore eases the pressure on your Web server and makes your applications run faster.

Secondly, Redirect maintains the original URL in the browser. This can really help streamline data entry techniques, although it may make for confusion when debugging.

IV.CHANNEL API FOR CHAT IMPLEMENTATION

The Channel API creates a persistent connection between your application and servers, allowing your application to send messages to JavaScript clients in real time without the use of polling. This is useful for applications designed to update users about new information immediately. Using the HttpRedirect, the OAuth 2.0 is implemented to make the cross domain call and channel API sends the data in a secured channel establish among the sender and receiver using MD5(Message Digest) Hashing, apart from this the channel API based multiuser chat is implemented in order to show the efficiency of web service call made on every chat. Each channel is subjected to be a separate network connection and help to access quickly to update the information to users.

ISSN: 2320-1363

The following elements are used mainly in channel API.

i. JavaScript Client - the user interacts with a JavaScript client built into a webpage.

ii. The Server - creating a unique channel for individual javascript clients and send a token. Receive the update messages from clients via HTTP request. Finally, sending update messages to clients via their channels.

iii. The client ID - The Client ID is responsible for identifying individual JavaScript clients on the server.

iv. Token - Tokens are responsible for allowing the JavaScript Client to connect and listen to the channel created for it.

v. The channel - A channel is a one-way communication path through which the server sends updates to a specific JavaScript client identified by its Client ID.

vi. The message - Messages are sent via HTTP requests from one client to the server.

vii. Socket - The JavaScript client opens a socket using the token provided by the server. It uses the socket to listen for updates on the channel.

viii. Presence - The server can register to receive a notification when a client connects to or disconnects from a channel.

The two diagrams illustrate the life of a typical example message sent via Channel API between two different clients using one possible implementation of Channel API.

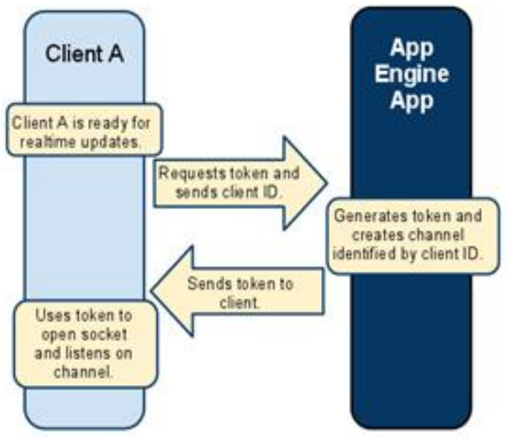
Client A
Client A is ready for realtime updates.
Requests token and sends client ID.
App Engine App
Generates token and creates channel identified by client ID.
Sends token to client.
Uses token to open socket and listens on channel.

Fig 3: Request token in channel API

This diagram shows the creation of a channel on the server. In this example, it shows the JavaScript client explicitly requests a token and sends its Client ID to the server. In contrast, you could choose to design your application to inject the token into the client before the page loads in the browser, or some other implementation if preferred.

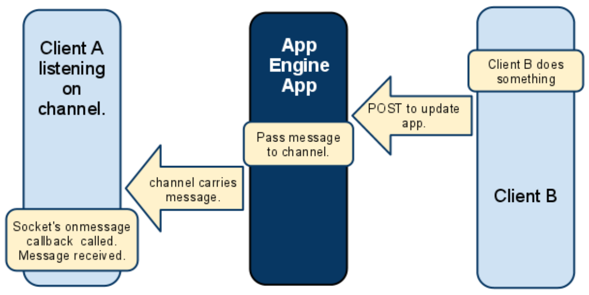

Fig 4: Posting message in channel API

Next, the server uses Client A's Client ID to create a channel and then sends the token for that channel back to Client A. Client A uses the token to open a socket and listen for updates on the channel.

V. CONCLUSION

In this paper, a new technique has been proposed to get optimized Web Service Composition. To achieve the Quality of Service, a CORS mechanism is used to make the cross domain call. Using HTTP Redirect, the fast data retrieval access can be obtained which causes high throughput and reliability. Also, to increase the efficiency of QoS, Channel API is used for multiuser chat where the web services call made on every chat also, it update the information quickly. For security reasons, Open Authentication protocol is used.

Thus our whole process satisfies the non-functional properties of QoS. Computational cost and time complexity is reduced by our algorithm called approximation algorithm. In a large web service repositories the above process are applicable to obtain the optimal QoS. Moreover, the overall global QoS constraints are highly optained.

REFERENCES

[1] Java, The Complete Reference, 8th Edition, by Herbert Schildt, Comprehensive Coverage of the Java Language.

[2]Professional JavaScript for Web Developers, by Nicholas C. Zakas, Third Edition.

[3]JavaScript and HTML5 Now, A New Paradigm for the Open Web, by O'Reilly, Kyle Simpson.

[4]http://www.kpmg.com/Global/en/IssuesAndInsights/ArticlesPublications/Documents/cloud-clarity.pdf (Accessed 2nd November 2014)

[5]Mell, P., and Grance, T., 'The NIST definition of cloud computing', 2011

[6]Gartner: 'Forecast Overview: Public Cloud Services, Worldwide, 2011-2016, 2Q12 Update', in Editor (Ed.)^(Eds.): 'Book Forecast Overview: Public Cloud Services, Worldwide, 2011-2016, 2Q12 Update' (2012, edn.), pp. 18-19 .

[7]http://ec.europa.eu/digital-agenda/events/cf/e2wp2014/document.cfm?doc_id=23902. (Accessed 18th November 2014)

[8]Gracia, J., and Mena, E., 'Semantic heterogeneity issues on the web', Internet Computing, IEEE, 2012, 16, (5), pp. 60-67

[9]Kobayashi, M., and Takeda, K., 'Information retrieval on the web', ACM Computing Surveys (CSUR), 2000, 32, (2), pp. 144-173

[10]Nagireddi, V.S.K., and Mishra, S., 'An ontology based cloud service generic search engine', in Editor (Eds.): 'Book An ontology based cloud service generic search engine' (IEEE, 2013, edn.), pp. 335-340 .