Securing application with Access Manager and enabling delegated authorization to third party vendor using OAuth

BITS ZG628T: Dissertation

Mid-semester Dissertation outline

by

Dennis Abraham

2016HT13037

Dissertation work carried out at
Infosys Ltd., Mysore

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE

PILANI (RAJASTHAN)

November 2018

Securing application with Access Manager and enabling delegated authorization to third party vendor using OAuth

BITS ZG628T: Dissertation

Mid-semester Dissertation outline

by

Dennis Abraham

2016HT13037

Dissertation work carried out at
Infosys Ltd., Mysore

Submitted in partial fulfilment of M.Tech. Software Systems degree programme

Under the Supervision of

Mahesh Shivananjappa

Infosys Ltd., Mysore

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE

PILANI (RAJASTHAN)

November 2018

ABSTRACT

BITS ID No.	**:** 2016HT13037
NAME OF THE STUDENT	**:** Dennis Abraham
EMAIL ADDRESS	**:** dennisabraham127@gmail.com
STUDENT'S EMPLOYING	
ORGANIZATION & LOCATION	**:** Infosys Ltd., Mysore
SUPERVISOR'S NAME	**:** Mahesh Shivananjappa
SUPERVISOR'S EMPLOYING	
ORGANIZATION & LOCATION	**:** Infosys Ltd., Mysore
SUPERVISOR'S EMAIL ADDRESS	**:** Mahesh_s08@infosys.com

DISSERTATION TITLE **:** Securing application with Access Manager and enabling delegated authorization to third party vendor using OAuth

ABSTRACT :

The main goal of this project is to implement OAuth 2.0 specification for third-party authorized access to secure data. OAuth is an industrial standard used by multiple web services to access private data of user, once user gives consent. This specification ensures that third-party client do not get access to user credentials or profile.

In this project, a rudimentary banking application is designed and implemented in Java 1.7. This is protected by Oracle Access Manager. Another application- aggregator, requests access to user bank info to show the user account information in one page. This flow of request, consent and access of private data (by aggregator) is handled by OAuth implementation in Java 7.

Broad Academic Area of Work : **Application and Network Security**

_____________________   _________________________

Signature of the Student	**Signature of the Supervisor**
Name: ______________	**Name: ________________**
Date:	**Date:**
Place:	**Place:**

Acknowledgement

I take this opportunity to owe a great many thanks to people who helped and supported me during the accomplishment of this dissertation.
I would like to extend my gratitude and thanks to Mr. Mahesh Shivananjappa, the supervisor of this project for guiding and helping me out technically during all the phases of this project with attention and care. He has helped me collecting necessary requirements and channelled my effort to get in touch with correct end-users and take inputs from them.
I express my deep sense of gratitude to Mr. Shinu Thomas, the additional examiner of this project for his timely support and technical guidance.
Thanks and appreciation to my colleagues - Bettes John and Biswajeet Chakraborty at DCSF project, Infosys Mysore for their support.
I would like to thank BITS, Pilani for providing me this great opportunity to explore in new areas.

Dennis Abraham

Date:

1. Introduction

1.1 Challenges

In the modern, present day scenario of the internet, almost each person has access to internet or is connected to cyber-world in some way. In India alone, we have approximately 340 million smartphone users, and 462 million internet users, just behind China [1].

As the web grows, more and more sites rely on distributed services and cloud computing: a photo lab printing your Flickr photos, a social network using your Google address book to look for friends, or a third-party application utilizing APIs from multiple services.

The problem is, for these applications to access user data on other sites, they ask for usernames and passwords. Not only does this require exposing user passwords to someone else — often the same passwords used for online banking and other sites — it also provides these applications unlimited access to do as they wish. They can do anything, including changing the passwords and lock users out.

There are some critical (in sense of security) applications like bank aggregators that used to show all investments, savings etc from multiple banks in their home page. Such applications, earlier (early 2000s) used to request user credentials to all their banks and investments, which will be used by the services to give all-in-one financial dashboard to the user. This practice was quite concerning- if the aggregator gets attacked, all the affected users' financial investment was at risk.

1.2 Solution

The solution is never to share user credentials with any services, but at the same time ensuring user be able to use the services. This was this need that introduced OAuth specification in 2010 and OAuth 2.0 was released in 2012 by IETF [2]. OAuth 2.0 is not back-compatible and is more secure and easier to implement than its predecessor [3]. OAuth is an authorization protocol.

OAuth defines four actors(entities) to handle the delegated authorization flow-

- **Resource Owner (RO):** The Resource Owner is the entity that controls the data being exposed by the API, and is, as the name suggests, the designated owner.
- **Authorization Server (AS):** The Security Token Service (STS) that issues, controls, and revokes tokens in the OAuth system. Also called the OAuth Server.
- **Client:** The application, web site, or other system that requests data on behalf of the resource owner.
- **Resource Server (RS):** The service that exposes and stores/sends the data; the RS is typically the API.

In the traditional client-server authentication model, the client uses its credentials to access its resources hosted by the server. OAuth introduces a third role to this model: the resource owner. In the OAuth model, the client (which is not the resource owner, but is acting on its behalf) requests access to resources controlled by the resource owner but hosted by the server.

For the client to access resources, it first must obtain permission from the resource owner. This permission is expressed in the form of a token and matching shared-secret. The purpose of the token is to make it unnecessary for the resource owner to share its credentials with the client. Unlike the resource owner credentials, tokens can be issued with a restricted scope and limited lifetime and revoked independently.

2. Implementation Overview

The implementation is categorized into four parts –

1. Designing and implementing Banking PoC application
2. Protecting bank application with Oracle Access Manager
3. Designing and implementing Vendor application (the aggregator)
4. Implementing OAuth flow

Below is a brief description of the four components-

2.1 Bank application

The bank application is a simple web application implemented in java 1.7 with user profile, account, branch and transactions. This application is implemented using Spring 2.0 API to implement the application as a web service. The bank details are stored in My SQL database to ensure that the Atomicity, Consistency, Isolation and Durability (ACID) of data is maintained. All the operations, except creation of schema is handled by the code.

The User Interface part is handled by Java Server Pages (JSP), which communicate to the business logic part of the coded using HTTP methods like GET and POST.

HTTP operation	**REST URL**	**Purpose**
PUT	/updateBalance?balance={balance}&accountNo={accountNo}	For money transaction
POST	/newUserEnrollment	For creating a new user
GET	/getAllusers	For Admin, to see all users
GET	/getMyInfo?accountNo={accountNo}	For user, to see his user information and account details

The JSP pages (except Login and error pages) and the REST URL will be protected by Oracle Access Manager.

2.2 Oracle Access Manager(OAM) protection

Oracle Access Manager is Oracle Identity Management's solution for web access management and user identity administration. Oracle Access Manager is designed to support complex, heterogeneous enterprise environments. As a key component of Oracle Fusion Middleware, it ensures ready support for Oracle's current and future ERP, CRM and Collaboration suite applications

Oracle Access Manager's Access System provides centralized authentication, authorization, and auditing to enable single sign-on and secure access across enterprise resources such as web and J2EE resources (JSP,servlets, EJBs, etc.) and legacy systems. The Access System is an extensible solution that can be leveraged to protect any kind of resources through policies. Legacy or custom applications can leverage its broad set of APIs to externalize authentication, authorization and auditing from their applications, and be able to enforce centrally managed access policies in their distributed applications or system.

To implement this protection, OAM needs following components [5]-

1. Applications
2. OAM Server and Oracle Access Management Console (installed on WebLogic AdminServer)
3. Policy Enforcement Agents
4. Credential Collectors and Communication Channels
5. SSO Engine
6. Access Policies
7. Policy Store
8. Cryptographic keys and Key Storage
9. Cookies

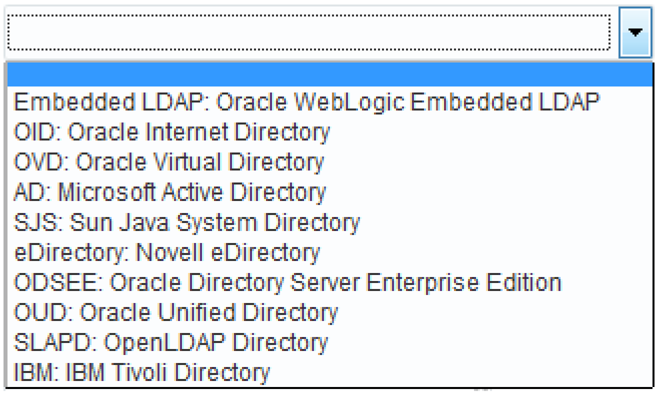
For installing these components, a cloud server with Windows Server 2012 R2 64-bit installed. Following components were installed-

a) Oracle 11g x64 database- to store the schema for OAM
b) Oracle Repository Creation Utility(RCU)- for creating schema in the above-mentioned database
c) Oracle WebLogic server 11g x64- it's is a application server on which Oracle Identity Suite products are installed
d) Oracle Access Manager 11g
e) Apache Tomcat Server- an application server for hosting web pages
f) Webgate- Oracle's Policy Enforcement Point (PEP) for intercepting all HTTP requests.

Note:- Although User profile do exist in the My SQL database, OAM's configuration of user profile is in an embedded LDAP server, which stores the users for OAM(admin users) as well. Since OAM provided LDAP authentication out-of-the-box, bank user username, password and accountNo will be stored in embedded LDAP and post authentication by OAM, OAM will set a

Header response containing accountNo, which will be passed to bank's /getMyInfo?accountNo={accountNo}, which loads the home page of that user.

Unfortunately, OAM does not support Database as a ID store [7]



Embedded LDAP: Oracle WebLogic Embedded LDAP
OID: Oracle Internet Directory
OVD: Oracle Virtual Directory
AD: Microsoft Active Directory
SJS: Sun Java System Directory
eDirectory: Novell eDirectory
ODSEE: Oracle Directory Server Enterprise Edition
OUD: Oracle Unified Directory
SLAPD: OpenLDAP Directory
IBM: IBM Tivoli Directory

Figure 1 Available ID stores for OAM

However, since the project intention is to implement OAuth 2.0 flow, the sync between Embedded LDAP user store and the bank database is done manually. The sync might be included in the final release.

2.3 Designing and implementing aggregator

Aggregator is an application that collects user's account details and provides a service based on the provided data. In financial domain, numerous third-party companies provide services like

- Insurance against a loan
- All-in-one dashboard to see all investments, bank accounts information in one page
- Calculating CIBIL score on basis of user account activities, etc

For example – Walnut, an app for smartphones is offers money management, loan remainders, bill reminders etc by linking to the user's bank without storing or requesting user credentials. Similarly, now popular Google's Tez enables user to do transactions without accessing bank application directly.

In this implementation, the aggregator will access user bank details, post user accent. This will also be developed in Java.

2.4 Implementing OAuth flow

OAuth 2.0 is implemented here with authorization code approach. The flow is briefed below-

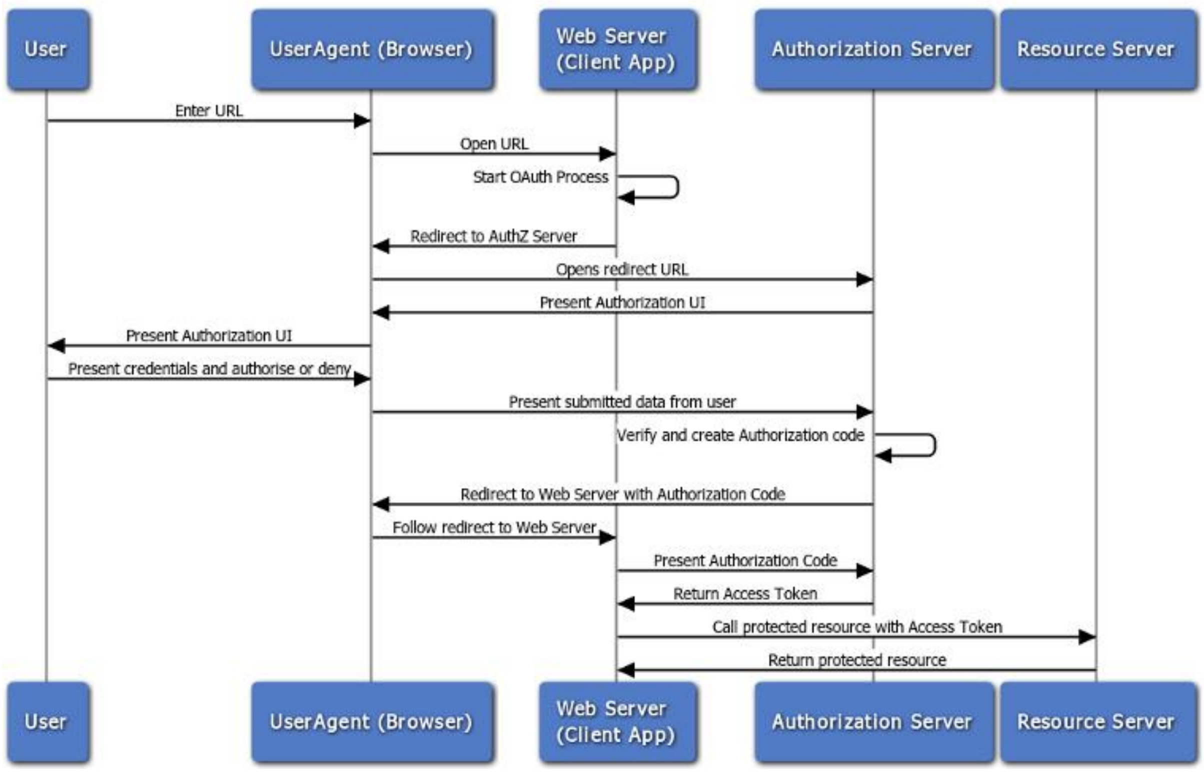

User
UserAgent (Browser)
Web Server (Client App)
Authorization Server
Resource Server
Enter URL
Open URL
Start OAuth Process
Redirect to AuthZ Server
Opens redirect URL
Present Authorization UI
Present Authorization UI
Present credentials and authorise or deny
Present submitted data from user
Verify and create Authorization code
Redirect to Web Server with Authorization Code
Follow redirect to Web Server
Present Authorization Code
Return Access Token
Call protected resource with Access Token
Return protected resource

Figure 2. Authorization code flow for OAuth 2.0

Here, the components are as follows-

a) User and browser, together act as Resource Owner
b) Web Server (Client App) is the aggregator
c) Authorization server is the OAM, since OAM provides the login flow to Bank application
d) Resource server is the bank application

The Authorization Code flow is as follows:

a. The Web server redirects the user to the API Gateway acting as an Authorization Server to authenticate and authorize the server to access data on their behalf.
b. After the user approves access, the Web server receives a callback with an authorization code.
c. After obtaining the authorization code, the Web server passes back the authorization code to obtain an access token response.
d. After validating the authorization code, the API Gateway passes back a token response to the Web server.
e. After the token is granted, the Web server accesses their data.

3. Status of the project

3.1 Bank Application status

The bank application, as explained in 2.1 is a java-based web-application that exposes the REST APIs for doing bank transactions.

Task ID	Task	Status
BANK_1	Develop user entities	COMPLETED
BANK_2	Link entities to Database using JPA implementation- Hibernate	COMPLETED
BANK_3	Implement relations and add constraints to entities	COMPLETED
BANK_4	Test bank operations like- view users, transactions, create new users using API testing application – POSTman	COMPLETED
BANK_5	Implement custom autogenerate accountNo when a new Account is created, so that the AccountNo is a 10-digit value	COMPLETED
BANK_6	Develop JSP pages to consume the REST URLs exposed by banking application	IN PROGRESS
BANK_7	Implement logging	PENDING

3.2 OAM protection status

OAM although is a powerful tool (hence quite heavy on CPU clocks) to protect web resources, however in this project, it will be implementing just the protection of banking resources. The check list for all the tasks are listed below-

Task ID	Task	Status
OAM_1	Create a Virtual machine on cloud	COMPLETED
OAM_2	Check the compatibility matrix of OAM components	COMPLETED
OAM_3	Setting up the VM with all required libraries like vcredist and java	COMPLETED
OAM_4	Installing OAM and its dependent modules	COMPLETED
OAM_5	Configure OAM and its components	COMPLETED
OAM_6	Verify if all services for OAM, is running	COMPLETED
OAM_7	Apply patches and fixes	COMPLETED
OAM_8	Verify if system can handle the load of OAM and its components	COMPLETED

OAM_9	Verify if webgate (Oracle's Policy Enforcement Point) is intercepting HTTP request	COMPLETED
OAM_10	Verify if OAM can authenticate the user	IN_PROGRESS
OAM_11	Populate headers with accountNo of user on login success	PENDING

3.3 Aggregator

Aggregator implementation is in progress.

Task ID	Task	Status
AGG_1	Design a login page and a landing page	PENDING
AGG_2	Protect the app using OAM, with a different user credentials (aggregator should not accept bank credentials)	PENDING
AGG_3	At landing page, user enters the AccountNo and is redirected to bank login page	PENDING

3.4 OAuth flow

OAuth flow is a bit complicated to implement. The status is as follows-

Task ID	Task	Status
OAUTH_1	Implement an authorization code flow in a PoC	COMPLETED
OAUTH_2	Is the client (PoC) redirects to PoC login page	COMPLETED
OAUTH_3	IS authorization code getting generated and is exchanged for access token(PoC)	COMPLETED
OAUTH_3	Integrate OAuth flow with banking application (to see user account details)	PENDING

Hence the PoC works, it just needs to be integrated with banking application.

3.5 Issues faced so far

1. OAM, although is a powerful designed by Oracle, a lot of things can go wrong even if it ticks all the right steps.
2. OAM and webgate had to be re-installed at-least four times, because there was no proper fix for the issues faced. The support could not assign my problem to top priority because it was not a production issue.
3. The machine configuration were edited- single core, high-ram to multi-core, less ram,
4. Although the compatibility matrix provided by Oracle were carefully followed, OAM 11g PS3 didn’t run properly, hence, OAM 11g PS2 was installed(which Oracle still supports)

Declaration by Student:

I certify that I have properly verified all the items in this checklist and ensure that the report is in proper format as specified in the course handout.

____

Place: ____________________________ **Signature of the Student**

Date: ______________________________

Name:_______________________________

ID No._______________________________