안전한 모바일 클라우드 컴퓨팅을 위한 ID 관리 시스템

Otieno Mark Brian, 이경현
부경대학교 IT 융합응용공학과
e-mail : mbotieno@gmail.com, khrhee@pknu.ac.kr

A Secure Identity Management System for Secure Mobile Cloud Computing

Otieno Mark Brian, Kyung-Hyune Rhee
Dept. of IT Convergence and Application Engineering, Pukyong National University

Abstract

Cloud computing is an up-and-coming paradigm shift transforming computing models from a technology to a utility. However, security concerns related to privacy, confidentiality and trust are among the issues that threaten the wide deployment of cloud computing. With the advancement of ubiquitous mobile-based clients, the ubiquity of the model suggests a higher integration in our day to day life and this leads to a rise in security issues. To strengthen the access control of cloud resources, most organizations are acquiring Identity Management Systems (IDM). This paper presents one of the most popular IDM systems, specifically OAuth, working in the scope of Mobile Cloud Computing which has many weaknesses in its protocol flow. OAuth is a Delegated Authorization protocol, and not an Authentication protocol and this is where the problem lies. This could lead to very poor security decisions around authentication when the basic OAuth flow is adhered to. OAuth provides an access token to a client, so that it can access a protected resource, based on the permission of the resource owner. Many researchers have opted to implement OpenID alongside OAuth so as to solve this problem. But OpenID similarly has several security flows. This paper presents scenarios of how insecure implementations of OAuth can be abused maliciously. We incorporate an authentication protocol to verify the identities before authorization is carried out.

1. Introduction

A new emerging trend in enterprise computing is the use of Smartphone devices. Smartphones have been advanced greatly, and so have malicious codes [1]. Smartphones are advancing in computational power; nonetheless, their major problem is resource poverty. Despite this, organizations are providing access to cloud services for their users with Smartphone-based clients [2, 3]. The location independence and computing power of a cloud joined with mobility of smartphone gives freedom of computing anything anywhere, resulting in a powerful ubiquitous computing model. This power and flexibility is bringing high popularity to Mobile Cloud Computing (MCC) [4, 5].

Organizations are hesitant to store and communicate valuable enterprise information to the third parties particularly due to the threat of unauthorized access to cloud. There have been proposal of novel authentication mechanisms such as the deployment of centralized Identity Management System specifically OAuth. OAuth is an open-web specification for organizations to access protected resources on one another's websites. It allows users to grant a third-party application access to their protected content without having to provide that application with their credentials.

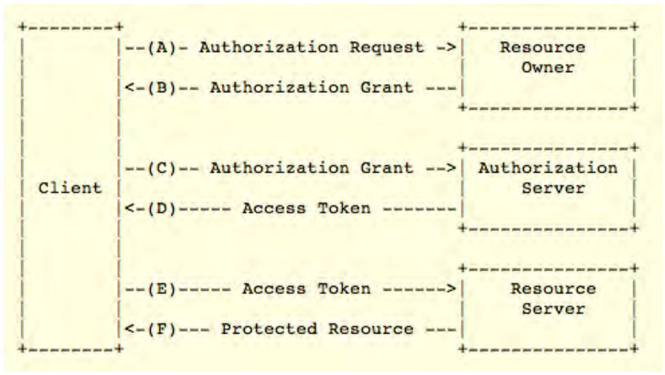
This paper is arranged as follows. In section I, we discuss an introduction of our study followed by a review of previous works in section II. In section III, we look at related works and formulate a problem domain in section IV. Section V covers the proposed method and countermeasures against attacks on users or applications that have implemented in this protocol. Finally, we conclude the paper in section VI.

2. Background

An identity management system manages the identities of individuals by ensuring their integrity throughout their lifecycle [6]. It also maintains the associated roles, access rights, authorizations, and privileges. In OAuth, client obtains a token the string denoting a specific scope and limited lifetime from authorization server to access a resource, hosted on resource server. OAuth consists of four modules (roles);

- Resource owner grants access to protected resource
- Resource server hosts protected resource
- Client is user/application that makes request to access resource on behalf of resource owner
- Authorization server issues token to client.

The communication flow of OAuth is as follows in the figure below:


Client
--(A)- Authorization Request ->
Resource Owner
<-(B)-- Authorization Grant ---
--(C)-- Authorization Grant -->
Authorization Server
<-(D)----- Access Token -------
--(E)----- Access Token ------>
Resource Server
<-(F)--- Protected Resource ---

(Figure 1)OAuth Communication Protocol

3. Related Work

Azeem *et* al [7] proposed a protocol to generate a distributed authorization token composed of two parts for a single resource access. First token is generated by IDM after producing credentials by the user, and is sent to user and cloud. Second token is generated by cloud and upon producing credentials by user, it is sent to user. Cloud also saves token for future use.

The sequence of action proposed is as follows:

1. The user logs into the cloud.
2. Cloud generates a token, and sends it to the user and saves it for future use. Cloud requests user to produce the token generated by IDM.
3. The user logs into the IDM.
4. IDM generates a token and sends it to both the user and the cloud.
5. User sends both tokens from cloud and IDM to cloud to request access.
6. Cloud compares token sent by user with tokens saved in its database.
7. Access is granted based on comparison results.

In this scenario, both cloud and user possess two tokens, while IDM server has access to a single token generated by it.

OpenID on the other hand presents a solution that allows organizing all online accounts under one login, using concept of Single Sign on (SSO) [8]. SSO authenticates users without having to keep track of usernames and passwords for all your site members. It’s also great for users, because they don’t have to create new credentials for every site.

4. Problem Domain

This section examines security challenges and insecure implementations associated with this protocol. OAuth does not provide native security nor does it guarantee privacy of protected data [9]. It has not authenticated the User yet it has delegated access to user’s information. It relies on implementers of OAuth, and protocols like SSL to protect exchange of data amongst parties. Thus, most security risks do not reside within the protocol itself, but rather its use.

Key Concepts

Server represents the Resource Provider while **User** represents the Resource Owner. **Client** represents the Consumer and **Client Credentials** are the consumer key and consumer secret to authenticate the Client. **Token Credentials** are access token and token secret used in place of User’s username and password.

The insecure storage of secrets is an area of concern especially two main areas; the shared secrets on the server and the consumer secrets on cross-platform clients.

To compute oauth_signature, Server must access shared-secrets (signed combination of consumer secret and token secret) in plaintext format as opposed to a hashed value. If Server and all its shared-secrets are compromised via physical access or engineering exploits, an attacker owns all credentials and can act on behalf of any Resource Owner of the compromised Server.

OAuth Clients use a combination of consumer key and secret to provide their authenticity to Server. This allows Clients to uniquely identify themselves to Server, giving Resource Provider ability to keep track of source of all requests. It also allows Server to let User know which Client application is attempting to gain access to their account and protected resource

Securing consumer secret on browser-based web application clients introduces same challenges as securing shared-secrets on Server. OAuth's dependency on browser-based authorization creates an inherit implementation problem for mobile applications not run in User's browser. Main concern is when implementers store and obfuscate the key/secret combination in Client application itself. This makes key-rotation almost impossible and enables unauthorized access to decompiled source code where consumer secret is stored.

The core function of consumer secret is to let Server know which Client is making request. So compromised consumer secret does not directly grant access to User's protected data. However, compromised consumer credentials could lead to some security threats:

- Server must keep track of all Clients and their authenticity, therefore safeguarding consumer credential is critical.
- Attacker uses compromised Client Credential to imitate valid Client and launch phishing attack by submitting request to Server on behalf of victim to access sensitive data.

Regardless, whenever consumer secret of mobile Client application is compromised, Server must revoke access for all users of the compromised Client application. Client must then register a new key which is a lengthy process, embed it into the application as part of a new release and deploy it to all its Users. This is a hectic process that could take weeks to restore service, and will surely impact the Client's credibility.

As regards OpenID, it uses a single sign-on protocol which permits users to sign in to a range of websites with their accounts [10]. This results in security flaws where scammers can be able to log into the websites as somebody else. With OAuth, there is naive expectation that access token is coming from resource owner. Hard reality is people go to questionable websites and use websites such as Facebook or Twitter to login. This avoids sharing their email and password with site. The problem is in the authentication case, Websites do have incentive to inappropriately reuse the access token. Token is no longer just for accessing protected resource; it now carries an implicit notion the possessor is the resource owner. So we end up in a situation where any site the user logs into with their Facebook account can impersonate that user at any other site that accepts Facebook logins [11].

There is nothing in the OAuth client-side flow that proves the issuer you sent the request to through the browser ever received it and is the one responding. Only the access_token parameter is generated by the Authorization server, all the other parameters are dropped. The threat arises because the client has no way to tell who the authorization server issued the access_token to.

5. Proposed Solution

Integrity protection of client credentials and token credentials works well when the tokens are stored on servers. Secrets can be isolated and stored in database or file-system with proper access control and file encryption.

Current OAuth mobile Client applications embed Client Credentials directly into application. Obfuscation is being employed as an alternative for secure implementation. Obfuscate consumer secret by splitting it into segments or shifting characters by an offset, then embed it in the application.

Implementing a better architectural concept requires some deviation from typical OAuth flow: The Service Provider could require Clients to bind their Consumer Credentials with a device-specific identifier.

Prior to initial OAuth handshake;

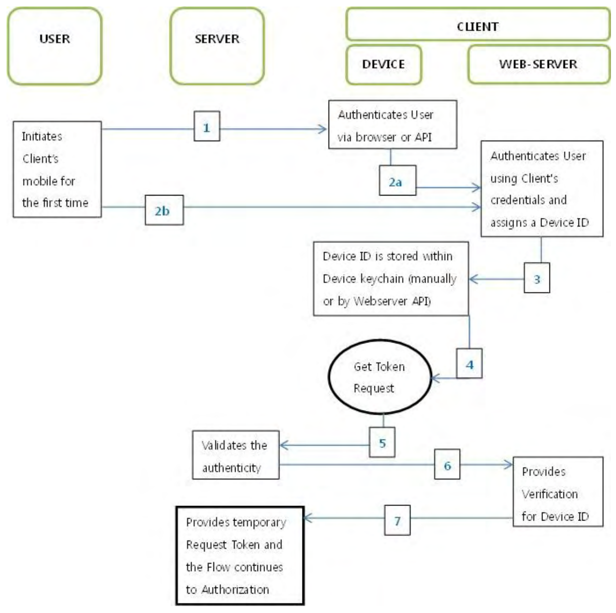
- Mobile application authenticates User to the Client application via username and password.
- Mobile Client then retrieves the Device ID from the Client's back-end server and stores it securely on the device itself.
- Once initial request is submitted to Server with both Client Credentials and Device ID, Service Provider validates authenticity of Device ID against Client's back-end server.

The proposed OAuth handshake and delegation workflow follows the steps as shown in figure 2.

1. User initiates Client's mobile
2. Client Device Authenticates the User via a browser or API
   User enters his Client's username and password to the Client's Web-server

3. The Web-server Authenticates User using Client's credentials and assigns a Device ID
   Device ID is stored within Device keychain (manually or by Webserver API)
4. The Device gets a Token Request
5. The Server validates the authenticity of the Tokens
6. The Web-server provides Verification for Device ID
7. The Server then provides temporary Request Token and the Flow continues to Authorization

OAuth’s strength is that it never exposes a User’s Server credentials to the Client application. Instead, it provides the Client application with temporary access authorization that User can revoke if necessary. So when Server is not sure of the authenticity of the Client’s key/secret, it will not entirely rely on these attributes to validate the Client.


USER
SERVER
CLIENT
DEVICE
WEB-SERVER
Initiates Client's mobile for the first time
1
Authenticates User via browser or API
2a
2b
Authenticates User using Client's credentials and assigns a Device ID
Device ID is stored within Device keychain (manually or by Webserver API)
3
4
Get Token Request
5
Validates the authenticity
6
Provides Verification for Device ID
7
Provides temporary Request Token and the Flow continues to Authorization

(Figure 2) OAuth Handshake and Delegation Workflow

6. Conclusion

As web grows, more sites rely on distributed services and cloud computing. With today's integrated web, users demand more functionality in terms of usability and cross platform integration. It's up to implementers and security professionals to safeguard user and also organizational data. Implementers should not rely solely on protocols only to provide all security measures. Implementers should instead be careful to consider all the avenues of attack exposed by an individual protocol, and therefore design the applications with the security risks factored in.

This paper presents OAuth protocol and outlines some of the security concerns related to its authentication mechanism. Despite being coupled with OpenID as an authentication protocol, there are still some security flaws. This paper therefore brings out the solution by proposing an alternative protocol to help in authentication by binding their consumer credentials with a device specific identifier.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education (NRF-2013R1A1A4A01009848)

References

[1] I. Bernik and B. Markelj, “Blended threats to mobile devices on the rise,” in *2012 International Conference on Information Society (i-Society)*, 2012, pp. 59 –64.
[2] “R. G. Chicone, An Exploration of Security Implementations for Mobile Wireless Software Applications within Organizations. Minneapolis: Graduate Faculty of the School of Business and Technology Management, Northcentral University, 2010.”
[3] “M. K. Riedy, S. Beros, and H. J. Wen, ‘Management Business Smart Phone Data’ in Journal of Internet Law, pp. 3-14, 2011.”
[4] F. Liu, P. Shu, H. Jin, L. Ding, J. Yu, D. Niu, and B. Li, “Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications,” *IEEE Wireless Communications*, vol. 20, no. 3, pp. 14–22, 2013.
[5] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, “Heterogeneity in mobile cloud computing: Taxonomy and open challenges,” *IEEE Communications Surveys Tutorials*, vol. Early Access Online, 2013.
[6] “Bishop, M., Computer Security: Art and Science, Reading, MA: Addison-Wesley Professional, 2002.”
[7] A. Ahmad, M. M. Hassan, A. Aziz, “A Multi-Token Authorization Strategy for Secure Mobile Cloud Computing,” 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, 2014.
[8] M. A. P. Leandro, T. J. Nascimento, D. Santos, D. R, C. M. Westphall, and C. B. Westphall, “Multi-Tenancy Authorization System with Federated Identity for Cloud-Based Environments Using Shibboleth,” presented at the ICN 2012, The Eleventh International Conference on Networks, 2012, pp. 88–93.
[9] D. H., “The OAuth 2.0 Authorization Framework.” [Online]: http://tools.ietf.org/html/draft-ietfoauth-v2-31.
[10] "http://openid.ne/.".
[11] K. Kiani, Four Attacks on OAuth – How to Secure Your OAuth Implementation, SAN Institute, 2011