International Journal of Emerging Trend in Engineering and Basic Sciences (IJEEBS)
ISSN (Online) 2349-6967
Volume 2 , Issue 1(Jan-Feb 2015), PP64-69

Security Assessment OAuth 2.0 System

Bharat Talaviya[1], Namrata Shroff[2]
CSE, Government Engineering College, Gandhinagar Sector 28, India
)

Abstract :- OAuth 2.0 System aims to unify the experience and implementation of delegated web service authentication into a single, community-driven protocol. The OAuth protocol enables web applications to access protected resources from a web service via an API, without requiring users to disclose their service provider credentials to the consumers. More generally, OAuth creates a freely-implementable and generic methodology for API authentication. In this work, we assess the different OAuth security approach and formalize the protocol using proof of possession architecture. The proof of possession gives some hope that the days of relying primarily on passwords and access tokens may be behind us within a few years.

Keyword :- Authorization, Authorization, Information Security, OAuth, Proof of Possession (PoP).

I.INTRODUCTION

Open APIs are one of the driving forces behind the modern web. Entire eco-systems of third party applications have grown around social media, content services and shopping websites resulting in more engaging and connected web experiences. In return, services with open APIs have benefitted by increases in the number of users and creating new revenue streams.

Developing and implementing best practice security solutions is not straightforward for the developers who make services and applications and numerous attempts have been tried to help users and services securely share data. API tokens, SAML and OpenID are some of the well-known solutions, however there is another solution, called OAuth (Open Authorization).

OAuth is a security protocol that enables users to grant third-party access to their web resources without sharing their authentication credentials with the third-party. Applications communicate with an authorization server to acquire an "access token", used to authenticate requests to an API server. [2] OAuth describes four players:

A. Client: Third-Party Application

The client is the application that is attempting to get access to the user's account. It needs to get permission from the user before it can do so.

B. Resource Server: API

The resource server is the API server used to access the user's information.

C. Resource Owner: User

The resource owner is the person who is giving access to some portion of their account.

D. Authorization Server: Often the same as the API server

The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

The tasks of resource owner and authorization owner players are performed by a single entity in most cases.

1.1 OAUTH PROTOCOL FLOW

Figure 1 outlines OAuth Protocol Flow procedure which is described as below: 1) Connect the Client to the Protected Resource. 2) End User initiates the Client action. 3) Client redirects User to Authorization Server. 4) User authenticates to Authorization Server. 5) User authorizes Client. 6) Authorization Server issues authorization code. 7) Authorization Server redirects User to Client. 8) Client sends code to Authorization Server. 9) Authorization Server issues token(s). 10) Client accesses Protected Resource.

1.2 OAUTH 2.O MODEL

A useful organizing model for thinking of OAuth 2.0 is: mechanisms by which a client can obtain a security token from an appropriate authority in order to use that token for authenticating a subsequent API call,

and mechanisms by which a client can present a security token as part of an API call in order to authenticate it (and thereby enable an authorization decision by the API hosting resource server).

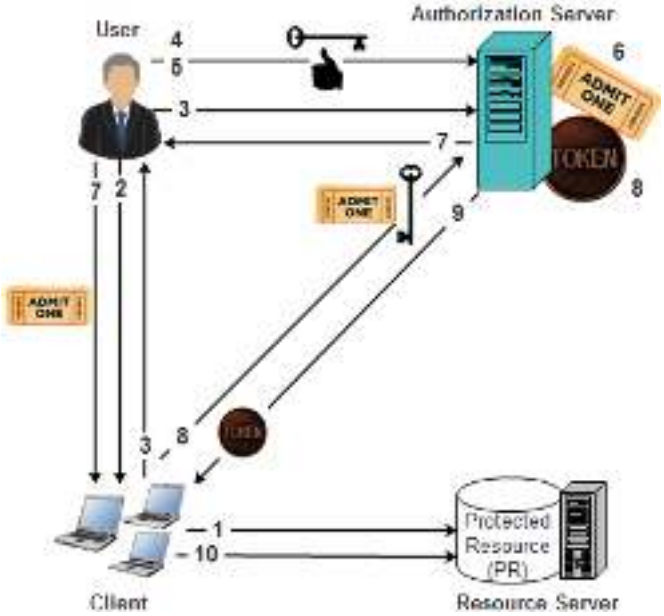

Fig 1: OAuth 2.0 Protocol Flow

II. SECURITY ASSESSMENT APPROACH

Several assessment approaches is used to examine the security of OAuth 2.0. To examine the Security of OAuth, OAuth Working group offers the Threat Model. Threat Model provides security considerations for OAuth that contains a comprehensive threat analysis and countermeasures for implementation developers [4]. Followings are assessment approaches:

A. Alloy Framework

Alloy Framework uses alloy specification language and alloy analyzer. OAuth model in Alloy Framework works as follow: First both the principals (participants) and values (unit of knowledge) are modeled as first class citizens i.e. both these sets of entities are represented using the sig (similar to class in OOL). Second, two abstract sig's principal and value are declared. Third, All principals Resource Owner, Resource Server, Client, Authorization Server extend the abstract sig principal. Here, the intruder is one or more of the four principals themselves, and intruder is not modeled as a separate individual. [5]

B. ProVerif Framework

ProVerif framework uses proverif specification language and follows pi-calculus. OAuth in ProVerif Framework consists of an unbounded number of users and servers. Each user browse any trusted or malicious website but only sends secret data to trusted sites. Following applications are hosted by server.

Login: An application models form based login and consists of two processes, LoginApp for server process and LoginUserAgent for user agent process.

DataServer: An application models resource servers and consists of two process, DataServerApp with two functions getData and storeData, and DataServerUserAgent for modeling the behavior of users.

OAuthAuthorization (User-Agent Flow): A third party application models user agent flow of OAuth protocol and consists of two processes, OAuthImplcitServerApp for authorization servers, and OauthUserAgent for resource owners.

OAuthAuthorization (Web Server Flow): A third party application models web server flow of OAuth protocol and consists of two processes, OAuthExplicitServerApp for authorization servers, and OauthExplicitClientApp for clients. [6]

C. ProVerif Framework with WebSpi Library

Figure 2 represents the WebSpi model. Users and Servers are the principals or agents of our model. Users contain credentials to authenticate with respect to a specific web application identified by a host name and

by a path in Credentials table. Servers contain private and public keys to implement TLS secure connections in server Identities table.

Table: Credentials (host, path, principal, id, secret)

Table: server Identities (host, principal, public, private)

Credentials and server Identities tables are private to the model and represent a preexisting distribution of secrets (passwords and keys). They are populated by the process Credential Factory that provides an API for the attacker (explained later) to create an arbitrary population of principals and compromise some of them. The process Web Surfer models a generic user principal that wants to browse a public URL. [6]

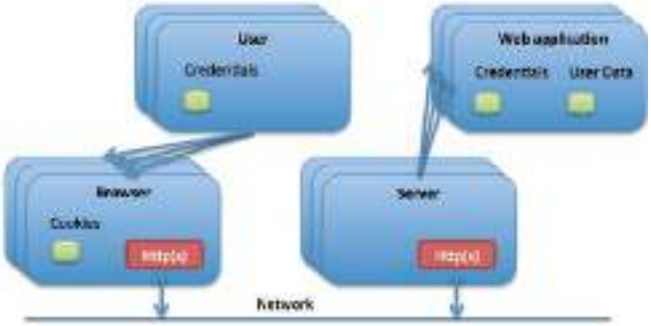

Fig 2: WebSpi Model

D. CryptoVerif Framework

CryptoVerif framework uses Blanchet Calculus for formalizing OAuth 2.0. In this framework, non-injective and injective correspondences are used to model the authentication from authorization server to end use and from authorization server to client. [7]

non-injective correspondence:

event authorization(x) □ user(x) ;to authenticate end-user by authorization server.

injective correspondence:

event authorization(x) □ client(x) ;to authenticate client by authorization server.

E. Universal Composability (UC) Security Framework

In this framework, an ideal third party functionality pass inputs by all protocol participants, and then computes the requisite function, and finally returns portions of computed function to each of the protocol participants. Security is approved by a simulation argument which essentially shows that if an Adversary can gain any information in the real-world implementation, it can also obtain the same information in the ideal third party based protocol. [8]

F. POAuth (Privacy Enhancing Open Authorization) Architecture

Trusted Computing is used to protect sensitive data and to secure cryptographic keys from eavesdropping. Trusted Computing Group (TCG) defines a hardware chip called Trusted Platform Module (TPM) with cryptographic mechanisms. TPM stored stores digests in shielded memory locations called Platform Configuration Registers (PCRs) using Secure Hash Algorithm (SHA-1). Upon first activation, the TPM generates a Storage Root Key (SRK). This non-migratable key is always present to secure underlying structure of TPM. In TPM, endorsement, storage, sealing, binding and attestation identity keys provide different security functionalities. The private part of the SRK itself never leaves the TPM and is thus protected by hardware.

Binding: In Trusted Computing context, associating data to be released only on a specific machine using a special type of asymmetric key, known as binding key, is termed as the concept of binding. In binding, a binding key (pair of public & private keys) is generated inside the TPM as non-migratable key. This key cannot be seen out of the TPM nor moved to any other TPM. Private part of binding key never leaves the TPM. The public part of the key can be broadcasted in a prescribed way for intended entities. To enable device specific authorization, we use the binding construct with trusted computing. The protocol is extended to device specific authorization to enhance privacy, called POAuth. There are three components of POAuth Architecture:

1) TPM enabled user device
2) POAuth enabled server
3) POAuth enabled consumer

III. LITERATURE REVIEW

OAuth 2.0 Protocol in Alloy Framework is not applicable at much lower layer of granularity. Future

task is to extend alloy framework at a lower layer of granularity. [5]

OAuth 2.0 Protocol using ProVerif Framework with WebSpi Library doesn't deal with XSS (Cross Site Scripting), SQL injection and DNS rebinding attacks. Future plan is to extend WebSpi library in order to capture these attacks and verify more web security mechanisms and protocols. [6]

OAuth 2.0 Protocol in CryptoVerif framework is proved not to guarantee authentication without secure measurement i.e. SSL protocol in computational model. Future work is the verification of the java implementation of OAuth 2.0 protocol in computational model. [7]

Universal Composability Security Framework spends a lot time to assess the security of the protocol even if the wrapper softwares are buggy, or downright maliciously collusive. Future work is to perform the Analysis of the Implicit Grant mode of OAuth 2.0 protocol. [8]

POAuth proposes an extension to the core OAuth protocol by introducing the concept of device-specific authorization. To ensure that the user can grant access to a consumer on specific device and novel technique using OAuth protocol in conjunction with hardware based security through the Trusted Platform Module. Future direction is performance analysis on a large scale web based service. [9]

IV. PROPOSED METHODOLOGY

Our analytic methodology is Proof of Possession (PoP) architecture to assess the security of OAuth 2.0 protocol. Proof-of-possession is a means of proving that a party sending a message is in possession of a particular cryptographic key. This is used as a proof that the correct party is sending the message, under the assumption that only that sender has possession the key. [10]

The proof of possession architecture assumes that the authorization server acts as a trusted third party that binds keys to access tokens. These keys are then used by the client to demonstrate the possession of the secret to the resource server when accessing the resource. The resource server, when receiving an access token, need to verify that the key used by the client matches the one included in the access token.

Figure 3 shows Interaction between Client and Authorization Server using Symmetric Key Procedure. In order to request an access token, the client interacts with the authorization server as part of the normal grant exchange. However, it needs to include additional information elements for use with the PoP security mechanism, as presented in message (1). In message (2), the authorization server then returns the requested access token. In addition to the access token itself, the symmetric key is communicated to the client.

Fig 3: Client ↔ Authorization Server using Symmetric

Fig 4: Client ↔ Authorization Server using Asymmetric

Figure 4 shows Interaction between Client and Authorization Server using Asymmetric Key Procedure.

If the client generates the key, pair it either includes a fingerprint of the public key or the public key in the request to the authorization server. The authorization server would include this fingerprint or public key in the confirmation claim inside the access token and thereby bind the asymmetric key pair to the token. If the client did not provide a fingerprint or a public key in the request then the authorization server is asked to create an ephemeral asymmetric key pair, binds the fingerprint of the public key to the access token, and returns the asymmetric key pair (public and private key) to the client.

Once the client has obtained the necessary access token and keying material it can start to interact with the resource server. To demonstrate possession of the key bound to the access token, it needs to apply this key to the request by computing a keyed message digest (i.e., a symmetric key-based cryptographic primitive) or a digital signature (i.e., an asymmetric cryptographic primitive). When the resource server receives the request it verifies it and decides whether access to the protected resource can be granted. This exchange is shown in Figure 5.

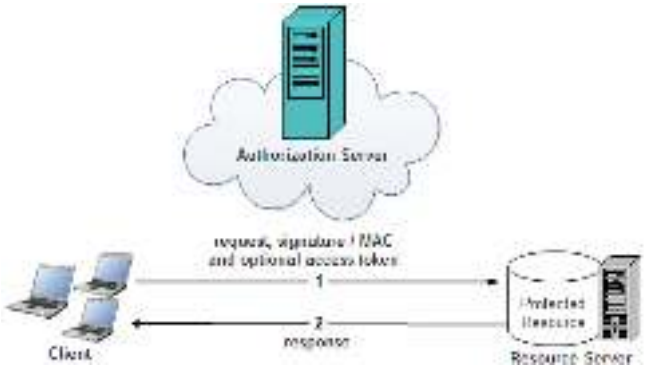

Fig 5: Client Demonstrated PoP

Fig 6: OAuth PoP Architecture

Access tokens are passed by value and allow the resource server to make authorization decisions immediately after verifying the request from the client. In some deployments, a real-time interaction between the authorization server and the resource server is envisioned that lowers the need to pass self-contained access tokens around. In that case, the access token merely serves as a handle or a reference to state stored at the authorization server. As a consequence, the resource server cannot autonomously make an authorization decision when receiving a request from a client but has to consult the authorization server. Figure 6 shows the OAuth 2.0 protocol interaction graphically.

V.CONCLUSION

OAuth is secure and efficient method for authorizing third party applications without releasing a user's access credentials. There are different security approaches available for verifying the OAuth protocol like Alloy framework, ProVerif framework, ProVerif Framework with WebSpi Library, CryptoVerif Framework,

Universal Composability (UC) Security Framework and POAuth (Privacy Enhancing Open Authorization) Architecture but Proof of possession architecture offer benefits: Preventing Access Token Re-Use by the Resource Server, TLS Channel Binding Support, Access to a Non-TLS Protected Resource and Application Layer End-to-End Security.

REFERENCES

Journal Papers:

[1] E. E. Hammer-Lahav. The oauth 2.0 authorization protocol draft-ietf-oauth-v2-13. [Online]. Available: http://tools.ietf.org/html/draftietf-oauth-v2-13

[2] By Jakob Jenkov, "Tutorial on OAuth 2.0". Available: http://tutorials.jenkov.com/oauth2/index.html

[3] T. Lodderstedt, M. McGloin, and P. Hunt. Oauth 2.0 threat model and security considerations. [Online]. Available: http://tools.ietf.org/html/draft-ietf-oauth-v2-threatmodel-01

[4] Suhas Pai, Yash Sharma, Sunil Kumar, Radhika M Pai and Sanjay Singh. "Formal Analysis of OAuth 2.0 using Alloy Framework" 2011 International Conference on Communication Systems and Network Technologies. *978-0-7695-4437-3/11, 2011 IEEE DOI 10.1109/CSNT.2011.141*

[5] Chetan Bansal, Karthikeyan Bhargavan and Sergio Maffeis. "Discovering Concrete Attacks on Website Authorization by Formal Analysis" 2012 IEEE 25th Computer Security Foundations Symposium, *2012 IEEE 10.1109/CSF.2012.27*

[6] Xingdong Xu, Leyuan Niu and Bo Meng. "Automatic Verification of Security Properties of OAuth 2.0 Protocol with CryproVerif in Computational Model" 2013 Asian Network for Scientific Information. *Information Technology Journal 12 ISSN 1812-5638 /DOI:10.3923/itj.2013.2273.2285*

[7] Suresh Chari, Charanjit Jutla and Arnab Roy. "Universally Composable Security Analysis of OAuth v2.0" *IBM T.J. Watson Research Center, Yorktown Heights, NY 10598*

[8] Mohammad Nauman, Sohail Khan, Abu Talib Othman, Shahr ulniza Musa and Najeeb Ur Rehman. "POAuth: Privacy-aware Open Authorization for Native Apps on Smartphone Platforms", *2012 ACM 978-1-4503-1172-4*

[9] Michael B. Jones, "The Increasing Importance of Proof-of-Possession to the Web" W3C Workshop on Authentication, Hardware Tokens and Beyond

Books:

[10] Ryan Boyd (O'Reilly) , "*Getting started with OAuth 2.0*"