An Authorization Scheme Concealing Client's Access from Authentication Server

Takamichi Saito
dept. Science and Engineering
Meiji University
Kawasaki, Japan
saito@cs.meiji.ac.jp

Yuta Tsunoda
dept. Science and Engineering
Graduate School of Meiji University
Kawasaki, Japan
ce66018@meiji.ac.jp

Daichi Miyata
dept. Science and Engineering
Graduate School of Meiji University
Kawasaki, Japan
ce56036@meiji.ac.jp

Ryohei Watanabe
dept. Science and Engineering
Graduate School of Meiji University
Kawasaki, Japan
ce66034@meiji.ac.jp

Yongyan Chen
Kunming University of Science and Technology
Kunming, China

***Abstract*—OAuth 2.0 and OpenID Connect are widely used authorization frameworks in the development of web applications. However, there are privacy concerns when using these frameworks because an authentication server can easily identify the URI of a website being accessed by a client. In this study, we proposed and implemented an authorization scheme that allows a client to conceal a URI from an authentication server when a web service detects the client's request.**

Keywords—OAuth 2.0, OpenID Connect, Authorization, Privacy, Federated ID

I. INTRODUCTION

As web technologies have developed, increasing numbers of web applications such as email and social networking services (SNSs) are provided as web services. The number of SNS users has been increasing each year in Japan. In 2014, of 9,941 million Internet users, approximately 60.23 million used SNS, i.e., 60% of Internet users were also SNS users. In particular, there were approximately 24 million Facebook users [2,4,5] and 1,950 million Twitter users [3].

A mechanism called social login enables a user to access other web services using his/her SNS account. Use of this facility has become widespread. A social login account is provided by a separate organization from the SNS provider. A major advantage of social login is that a user is not required to remember a new ID and password, which simplifies the procedure for account creation in web applications. According to Janrain [6], 88% of users who had never visited a site selected to use social login, and 51% of Internet users are now using social login to access other web applications [7]. Social login is a form of single sign-on in which a user's information, e.g., name or age, is shared with a web application. Many social logins currently being implemented depend on protocols such as OAuth 2.0 [8] or OpenID Connect [9].

Because attribute information needs to be exchanged between an authentication server and a user, in OAuth 2.0 and OpenID Connect, a web application server will directly communicate with an authentication server or will redirect the web browser to the authentication server. Although a web application server and authentication server are in different domains or different carriers, the authentication server can collect information such as the URI of a web application that a user is visiting. This allows the authentication server to identify the web application and therefore to track and identify the terminal or web browser when a client requests a web application server. This raises privacy concerns. It is accepted that information about individuals who access a web application should be protected and not be unnecessarily disclosed even to concerned parties [10]. In this study, we proposed and implemented an approach to conceal a web application from an authentication server while providing a user's attribute data to the web application.

II. RELATED TECHNOLOGY

In this section, we discuss relevant technologies focusing on the method by which an authentication server traces a web service when a user visits.

A. OAuth 2.0

The OAuth 2.0 authorization framework allows a third-party application to obtain limited access to an HTTP service. OAuth 2.0 was developed in 2012 as RFC6749.

1) Terminology

- Authorization Server

  A server that authenticates end users and issues an authorization code and/or access token.

- Resource Server

  A resource server manages the protected resources of an end user and provides them to a client.

978-1-5090-0984-8/16 $31.00 © 2016 IEEE
DOI 10.1109/IMIS.2016.110

- Client

  An application provides services to the end users. An application requests end users' protected resource to a Resource Server.

- End User

  An end user is an owner of resources that are managed by a resource server. An end user has the authority to access protected resources and uses the services of a client.

- Authorization Code

  An authorization code is a character string that will be issued to a client from an authorization server when an end user passes the access authority to the client.

- Access Token

  An access token is a character string that is issued when a client has presented an authorization code to an authorization server.

2) Authorization Code Grant

Four flows are defined in OAuth 2.0. In this study, we discuss the authorization code grant. A client obtains a client_id from an authorization server before starting the authorization step. This is an identifier of the client. The procedure for client_id assignment is that clients register their identification information (redirect_uri etc.) on an authorization server. The redirect_uri is the client URL that receives the authorization code and access token.

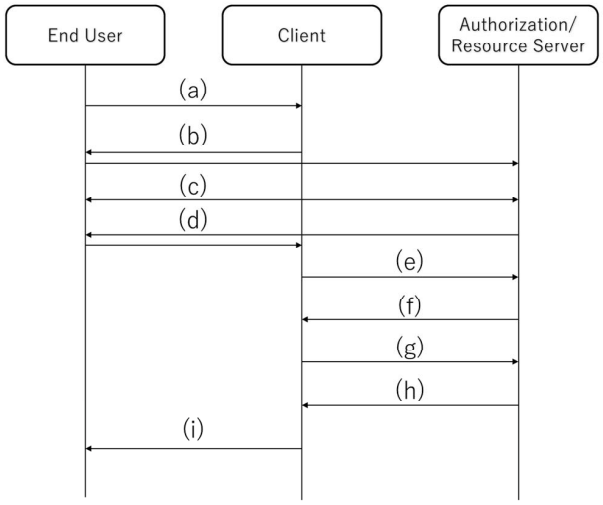

Figure. 1 Flow of OAuth2.0

a) End user accesses a client.

b) Client sends an authorization request to an authorization server to redirect the web browser. This request includes the redirect_uri, which has been registered on the authorization server in advance. In this case, the redirect_uri indicates the client's location. In addition, the scope encapsulated in the authorization request is a string of characters used to show the range of access authority that the client requests. The scope is pre-defined in the authorization server. For example, there is a case in which the scope is "name" and the authorization server specifies that the client requests the authority to obtain the end user's name.

c) In this step, the authorization server authenticates the end user. Note that the method of authentication is not within the scope of OAuth 2.0: password authentication is required in most cases. The authorization server usually demands access authentication information without seeking the user's agreement. Simultaneously, the authorization server displays a selection of access authorities specified in the authorization request in (b). The end user then selects one of them.

d) The authorization server sends the response to the redirect_uri specified in (b) by redirecting. The response includes the authorization code.

e) Using the POST method, the client sends the authorization code to obtain an access token.

f) The authorization server verifies the validity of the authorization code. If it is valid, the authorization server issues the access token and sends it to the client.

g) The client submits the access token to the resource server, which uses the authority specified in the token.

h) The resource server verifies the validity of the access token. If it is valid, the resource server protects resources specified in the access token.

B. OpenID Connect

OAuth 2.0 is an authorization framework that enables applications to obtain limited access to user accounts. OpenID Connect is a function of OAuth 2.0 that allows computing clients to verify the identity of an end user as well as to obtain authentication information about them. OpenID Connect was approved in February 2014 through the United States OpenID Foundation [11] as an API standard specification.

1) Terminology

- OpenID Provider (OP)

  An OP is a server that authenticates an end user and manages his/her account.

- Relying Party (RP)

  An RP is an application that entrusts the end-user authentication of the OP and provides services for the end user.

- End User

  An end user is a user to be authenticated to use the RP. The end user accesses the RP/OP through a web browser.

- ID Token

  An ID Token is an object encapsulating the security identity of an end user and includes an authentication

time. ID Token is generally described in the JSON Web Token [13].

2) Authorization Code Flow

Three flows are defined in OpenID Connect. In this study, the authorization code flow (Figure 2) is described as follows:

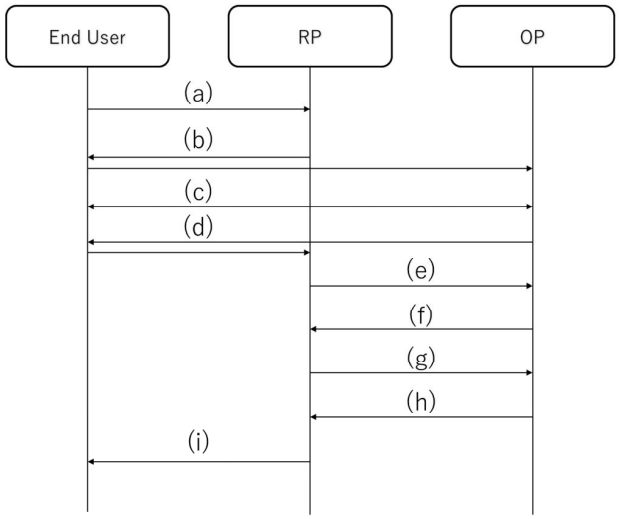

Figure. 2 Flow of OpenID Connect

a) End user accesses the RP to use a service.

b) RP submits an authorization request to the OP to redirect the web browser. The authorization request contains client_id, scope, and redirect_uri registered on an authorization server in advance. In this case, the redirect_uri allows the user to directly access the RP.

c) OP authenticates the end user. Note that the method of authentication is not within the scope of OpenID Connect.

d) OP sends the response to the redirect_uri specified in (b) by redirecting. The response includes the authorization code that was issued by the OP.

e) Using the POST method, the RP sends the authorization code to the OP to obtain the access and id tokens.

f) OP verifies the validity of the authorization code. If it is valid, the OP issues the access and id tokens and sends them to the client.

g) RP submits the access and id tokens to the OP to use the authority specified in the access token.

h) OP verifies the validity of the access and id tokens. If they are valid, the OP protects resources specified in the access token.

C. Problems with existing technology

In OAuth 2.0, when a client receives an authorization request (Figure 1 (b)), an authorization server obtains the identifier of the client from an end user. The authorization server also acquires the URI of the client; thus, there is a possibility of being able to determine what type of service the end user is accessing. The information shared between the authentication server and web application encapsulated in HTTP is therefore redirected to the URI. This is called the POST method, which can determine the URI of a client by inspecting to the referrer of the HTTP header.

From the above discussion, it can be observed that an authorization server and the OP can detect the web server that an end user is accessing in both OAuth 2.0 and OpenID Connect.

III. Proposed Scheme

A. Overview

To solve the problems raised in Section 2.3, we propose a new security authentication procedure. To conceal the URI information of a web server from an authentication server, the attribute information of an end user is treated as part of the authentication information and is saved in the authentication server.

In this scheme, using a browser extension of Google Chrome [14], the referrer information is filtered from the application server. The end user then interacts with the authentication server and access server through the browser extension. The entire procedure hides information about the web application from the authentication server.

B. Terms used in the proposed scheme

- Identity Provider (IdP)

  An IdP authenticates an end user and manages the end user's account. The IdP provides the end user's attribute information to the RP.

- RP

  A RP is a server that delegates user authentication to the IdP. The RP registers the URI of the IdP and the public key of the IdP in pairs and in advance.

- End User

  An end user is authenticated by the IdP and uses the RP. The end user accesses the RP and IdP through a web browser.

- Endpoint

  An endpoint is an RP's URL that accepts an authentication response message from the IdP.

- Nonce

  A nonce is an arbitrary string that ensures that the token is different for different sessions.

- Timestamp

  A timestamp identifies the time at which the token was issued.

- Scope

  A scope is a string that shows the range of the attribute information that a client requests. The scope is defined in advance by the IdP.

- Key

  This is a public key generated by the RP for each access by the end user. The key is used when IdP encrypts the attribute information of the end user. In contrast to a public key, a private key is held by the RP and should not be shared or distributed.

- Token

  A token is a hashed value of the endpoint, nonce, timestamp, scope, and key and is defined as follows:

  Token = H(Endpoint||Nonce||Timestamp||Scope||Key),

  where H is a hash function and || is a concatenation of strings.

C. *Flow of proposed scheme*

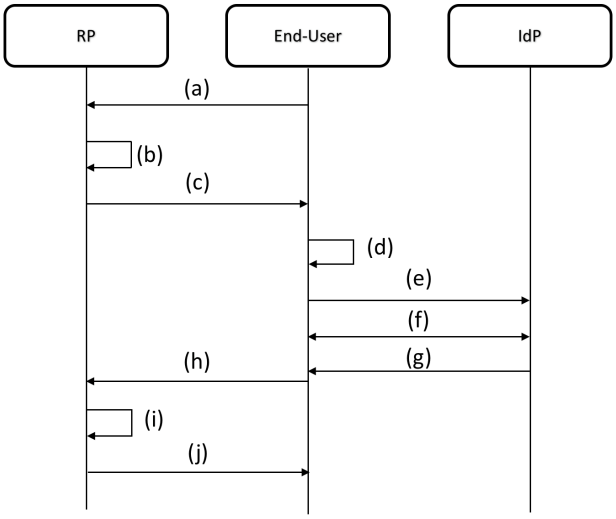

Figure. 3 Flow of proposed scheme

a) End user accesses the RP to use the RP's service.

b) RP calculates a token from an endpoint, nonce, timestamp, scope, and key as an authentication request.

c) RP sends the token, endpoint, nonce, timestamp, scope, and key to the end user.

d) End user verifies that the token and a hashed value of the concatenation of the endpoint, nonce, timestamp, scope, and key are equal (hereafter called token verification) and that RP and endpoint domains are equal (hereafter called endpoint verification). If these verifications fail, a flow is stopped.

e) End user sends the token, timestamp, scope, and key to the IdP as an authentication request.

f) IdP authenticates the end user. Note that the method of authentication is not within the scope of the proposed scheme.

g) IdP encrypts the attribute data with the key. The IdP sends a signed authentication response containing the token, timestamp, encrypted attribute data, and URL of the IdP to the end user.

h) End user sends the authentication response to the endpoint.

i) RP verifies the authentication response as follows:

1.Token is issued in (b).
2.Token has not been previously used.
3.Token has not expired.
4.Signature is valid.

If these conditions are satisfied, the RP decodes the encrypted attribute data.

j) RP provides the requested service, having confirmed that the IdP has authenticated the end user.

D. *Concealment from IdP*

In this section, we describe how the proposed scheme conceals information about the RP from the IdP. The RP has to transfer the parameters, i.e., the token, timestamp, scope, and key, to the IdP. Because the token is a hashed value, it is difficult for the IdP to identify the RP from the token. Because the scope is a character string that shows the range of attribute information that a client requests, the IdP cannot identify the RP from the scope. Because the timestamp is the time that the token was published, the IdP cannot identify the RP from the timestamp. Because the key is generated at each client access, the IdP cannot identify the RP from the key. It follows that the IdP cannot identify the RP from parameters sent from the IdP. This is shown in Figure. 3 (e).

IV. IMPLEMENTATION

In this section, we describe our implementation of the proposed scheme.

A. *Environment*

- IdP

OS: CentOS 6.6 (kernel2.632-504.el6.x86_64)
HTTP Server: Apache HTTP Server 2.2.15
Server Side Script: PHP 5.3.3
Database Server: MySQL Server 5.1.73

- RP

OS: CentOS 6.6 (kernel2.632-504.el6.x86_64)
HTTP Server: Apache HTTP Server 2.2.15
Server Side Script: PHP 5.3.3
Database Server: MySQL Server 5.1.73

- End User

OS: Microsoft Windows7 Home Premium
Web Browser: Google Chrome 48.0.2564.8

1) *Browser extension*

We wrote a browser extension for Google Chrome for relay communication with the IdP and RP. A browser extension is a program that is written in HTML, JavaScript,

and CSS. The browser extension created for this study executed the process shown in Figures 3 (d)–(h).

B. Implementation of browser extension

An endpoint, nonce, timestamp, scope, key, and token were generated and displayed when an end user accessed the RP (shown in Figure 3 (c)). The browser extension read the endpoint, nonce, timestamp, scope, key, and token on the webpage when the user clicked the icon of the browser extension. The browser extension then verified the endpoint and token (shown in Figure 3 (d)). This was implemented to determine the IdP by entering the end user's URI of the IdP. The end user checked the "Agree" box to pass the attribute data to the RP and clicked the "Submit" button. The RP send Token, Timestamp, Scope, and Key to the RP by using XMLHttpRequest in JavaScript [15]. Following submission, the end user was authenticated at the page of the IdP. After the user was authenticated, the IdP sent an authentication response to the browser extension. The browser extension sent the authentication response to the endpoint of the RP using the POST method. The assignment was then completed.

V. DISCUSSION

In this section, we present a discussion of the proposed scheme.

A. Comparison with related technology

To safeguard a user's privacy, the proposed mechanism communicates using a browser ID. An end user can log in without the knowledge of the web application being passed to an authentication server [16]. However, a problem exists within the browser ID that compromises the privacy of a user [17].

Previous studies of security authentication schemes focused on secure, privacy-respecting single sign-on systems (SPRESSOs) [18] that conceal the RP from the IdP using the server. These use a method called FWD that forwards messages to the IdP and RP. In contrast with SPRESSO, a browser extension was made responsible for forwarding information. In SPRESSO, the IdP can collect information on the RP that an end user has accessed by combining the IdP and FWD. To defend against such an attack, it is necessary for the RP or end user to check the behavior of the FWD. However, it is not easy to check the behavior because the FWD does not operate on the client side. In contrast, the proposed scheme makes it possible to verify the behavior of the browser extension before the flow starts because the end user or RP can view the source code of the browser extension.

Although a scheme using a browser extension to conceal a web service from an authentication server has been introduced in a previous study [19], this approach did not provide attribute data on an end user to the IdP or RP. The RP was therefore unable to provide a service to each end user. In contrast with the approach in [19], the RP in the proposed scheme solved this problem by providing the attribute data on an end user while concealing the RP from the IdP.

B. Security considerations

The proposed scheme uses TLS as the cryptographic protocol for providing secure communications. Unfortunately, when using TLS, a man-in-the-middle attack becomes possible using a malicious RP. In this section, we describe four cases of such an attack and demonstrate countermeasures provided by the proposed scheme.

1) Sending the token as it is

An attacker submits authentication data (token, nonce, endpoint, and so on) generated by the legitimate RP shown in Figure 4 (c) to an end user without rewriting. In this case, the verification of the endpoint makes it possible to detect that the sender of the authentication data was not authentic. This is shown in Figure 4 (g)

2) Rewriting of the endpoint

When an attacker attempts to send an authentication response to a malicious RP, the attacker will attempt to rewrite the endpoint, as shown in Figure 4 (e). In this case, the validation of the token makes it possible to detect the rewriting.

3) Rewriting of the endpoint and token

To bypass validation, an attacker rewrites the endpoint, as shown in Figure 4 (g), creates a fake token, and replaces the token generated by the legitimate RP. In this case, although it is possible to break through the verification (as shown in Figure 4 (g)), the attack is detected by the verification of the authentication response, as shown in Figure 4 (m). Because the attribute data of the end user is encrypted with the public key of the legitimate RP, a malicious RP cannot decrypt it.

4) Temporary replacement of the token

To bypass the validation of the token, as shown in Figure 4 (m), an attacker rewrites the endpoint and token and obtains the authentication response. When sending to the legitimate RP, the attacker replaces the fake token with the token that was generated by the legitimate RP. In this case, the digital signature of the authentication response is signed by the fake token, and the attack can be detected when the verification of the digital signature is conducted, as shown in Figure 4 (m).

C. Generation time of authentication data

In the proposed scheme, a private key, token, and public key are generated at each access of an end user, as shown in Figure 3 (b). We measured the execution time for generating the key and calculating the token 1,000 times. The evaluation environment comprised an Intel (R) Core (TM) i7-3610QM CPU @ 2.30 GHz. The generated key was of the RSA type, with a length of 2,048 bits.

The time taken to generate the Key and calculate the token 1,000 times was 2 min 3.0925 s. The time taken to generate a single key and calculate a single token was 123.0925 ms.

VI. CONCLUSIONS

In this study, we proposed a new authentication scheme based on OAuth 2.0 and OpenID Connect for hiding web access information from an authentication server. In the proposed scheme, a novel protocol provides authentication

information and attribute information. The proposed scheme was tested, and its performance was confirmed.

ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number 26330162. We are deeply grateful to Y. Iso and T. Watanabe for this work.

REFERANCES

[1] http://ictr.co.jp/report/20150729000088-2.html
[2] "Twitter" https://twitter.com/ http://openid.net/specs/openid-connect-core-1_0. html
[10] Yasuo Okabe, Hiroyuki Sato, Takeshi Nishimura, Kazutsuna Yamaji, Motonori Nakamura "Pseudonymized Proxies Which Conceal an Attribute Provider and a Service Providers Each Other" (DICOMO2013)
[11] "OpenID Foundation" http://openid.net/foundation/
[12] "The OpenID Foundation Launches the OpenID Connect Standard" http://openid.net/2014/02/26/the-openid-foundation-launches-the-openid-connect-standard/
[13] "JSON Web Token" https://tools.ietf.org/html/rfc7519
[14] "What are extensions?" https://developer.chrome.com/extensions
[3] "Facebook" https://www.facebook.com/
[4] http://www.cereja.co.jp/press_release20141212.pdf
[5] eMarketer "Asia-Pacific Grabs Largest Twitter User Share Worldwide" http://www.emarketer.com/Article/Asia-Pacific-Grabs-Largest-Twitter-User-Share-Worldwide/1010905
[6] "Janrain" http://janrain.com/
[7] 2014 Janrain US Consumer Research "Social Login and Personalization" http://www1.janrain.com/rs/janrain/images/Industry-Research-Social-Login-and-Personalization-2014.pdf
[8] "The OAuth 2.0 Authorization Framework" https://tools.ietf.org/html/rfc6749
[9] "OpenID Connect Core 1.0 incorporating errata set 1"
[15] XMLHttpRequest Level 1 W3C Working Draft 30 January 2014 http://www.w3.org/TR/2014/WD-XMLHttpRequest-20140130/
[16] Mozilla Persona https://login.persona.org/
[17] Daniel Fett, Ralf Küsters, Guido Schmitz "Analyzing the BrowserID SSO System with Primary Identity Providers Using an Expressive Model of the Web"
[18] Daniel Fett, Ralf Küsters, Guido Schmitz "SPRESSO: A Secure, Privacy-Respecting Single Sign-On System for the Web" Computer and Communications Security 2015
[19] Yuto Iso, Takamichi Saito "A Proposal and Implementation of an ID Federation that Conceals a Web Service from Authentication Server" The 29th IEEE International Conference on Advanced Information Networking and Applications (AINA-2015)

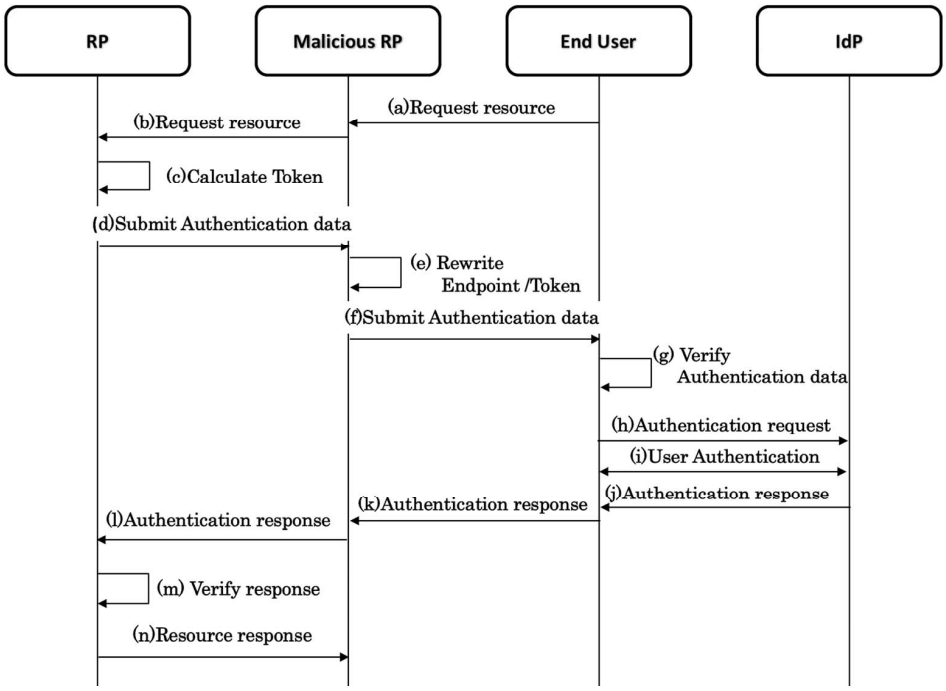

Figure. 4 Man-in-the-middle attack