XHR2 を用いたページ遷移を伴わない OAuth2.0 の実現方式の提案と実装

後藤 浩行† 村上 智佑† 齋藤 孝道‡

†明治大学大学院
‡明治大学

あらまし ユーザが，あるWebサービスにおける権限をマッシュアップサービスなどに委譲する際，OAuthでは ，アカウントを管理するサービスのWebページにリダイレクトする．リダイレクトにはページのロード及び，再描画，クライアントサイドアプリケーションの再実行というオーバーヘッドが発生する．そのオーバーヘッドを削減するため，本論文では，XHR2及びHTTP認証を利用することでクロスドメイン通信における安全な認証を可能とした，ページ遷移を伴わないOAuth2.0の実装を示す．

A Proposal and Implementation of OAuth2.0 using XHR2 without Redirection

Hiroyuki Goto† sTomosuke Murakami† Takamichi Saito‡

†Graduate School of Meiji University
‡Meiji University

Abstract In OAuth2.0, when a user delegates his authority in a web service to mashup service, his browser is redirected to a web page of a service managing his account. Redirection has a cost of the time in loading a page, redrawing a screen, and rerunning a client side program. In the paper, we implemented OAuth2.0 without redirection by using XHR2 and Basic authentication.

1 はじめに

インターネットの普及とともに Web を通してサービスを提供する Web アプリケーションが広く普及している．更に，近年では，運営主体の異なる複数の Web アプリケーションを組み合わせてサービスを提供するマッシュアップサービスが行われるようになった．従来，Web アプリケーションがそれぞれ保持しているエンドユーザ情報を，エンドユーザに代わり，マッシュアップサービスで利用する場合，マッシュアップサービス側において，エンドユーザの ID とパスワードを保持する運用形態が多かった．それに対し，エンドユーザの持つ情報にアクセスする権限を，マッシュアップサービスなどを含めた Web アプリケーションに安全に委譲する仕組みとして OAuth[1]が提案された．OAuth では，マッシュアップサービス等にエンドユーザが権限の委譲を行う際，エンドユーザは権限を管理している Web アプリケーションのページにリダイレクトされる．このような一般的な OAuth の実装の利用では，ブラウザにおいてリダイレクトによるページ遷移はページのロード，再描画，クライアントサイドアプリケーションの再実行というオーバーヘッドが伴う．また，クライアントサイドアプリケーションは処理が中断されるため，中断された状態より処理を再開したい場合は状態管理を行う必要があり，開発コストは増加するケースも想定される．

本論文では，XHR2（XMLHttpRequest Level2）[2]および HTTP 認証[3]を用いて，ページ遷移を伴わない OAuth2.0 の実現方式の提案とその実装を示す．

2　OAuth

２．１　概要

OAuth とはエンドユーザの権限の委譲を実現するプロトコルである．OAuth2.0（以降，OAuth1.0a と区別しない場合は OAuth と呼ぶ）では，エンドユーザの ID，パスワードといったクレデンシャル情報を保持する Web アプリケーションが，エンドユーザの同意のもとで，マッシュアップサービスなどを提供する Web アプリケーションにエンドユーザのクレデンシャル情報を渡すこと無く，権限を委譲することが可能である．

２．２　関連用語

ここでは，OAuth に関する用語を説明する．

- AuthorizationServer

  後述の EndUser の承認があった際，後述の Client に後述のアクセストークンの発行及びその管理を行う．

- ResourceServer

  後述の EndUser の保護されたリソースを保持している．EndUser が後述の Client に権限を委譲した後，Client から提出されるアクセストークンの検証を行い，Client に対して保護されたリソースの提供を行う．

- アクセストークン

  アクセストークンは，作成時に都度生成されるランダムな文字列により特定される情報であり，AuthorizationServer に保存される．作成の際，権限の委譲範囲を示す scope，有効期限を示す expires と任意のオプションを合わせて保存される．

- EndUser

  ResourceServer の持つ保護されたリソースへのアクセス権限を持ち，Client に保護されたリソースへのアクセスを許可する．

- Client

  EndUser に権限を委譲してもらうために，AuthorizationServer にアクセストークンの要求をする．そのアクセストークンに対応するリソースへアクセスし，EndUser に Web サービスを提供する．事前に Client はサービス名や EndUser が認可を行った後にリダイレクトされる Client の URL（以降 redirect_url と呼ぶ）などの情報を AuthorizationServer に登録する．その際，AuthorizationServer が Client を一意に識別するため，client_id と，AuthorizationServer と共有される秘密値である client_secret が発行される．

２．３　フローの説明.

OAuth2.0 は現在 IETF により仕様策定中である．draft10[1]ではフローが複数定義されているが，本論文で利用した WebServer フローについて図 1 中の番号と対応させて説明する．

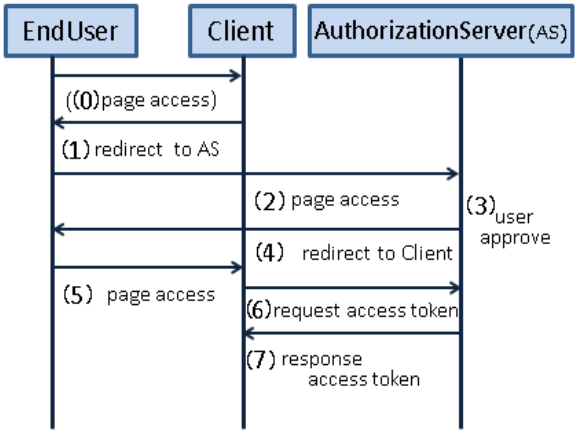

図 1.　OAuth2.0 のシーケンス

(0) OAuth の範囲外なので，仕様では定義されていない．EndUser が Client にアクセスする．これを契機に Client は OAuth で権限の委譲を要求する．

(1) Client により EndUser は AuthorizationServer の認可用ページにリダイレクトされる．この際，リダイレクト先の URL には client_id と scope 等がパラメータとして付加される．

(2) EndUser は Client からのリダイレクト指示により AuthorizationServe の認可用ページにアクセスさせられる．AuthorizationServer は EndUser を特定する必要があるため，この段階，もしくはそれ以前に認証処理を行う必要がある．ここでの EndUser の認証に関しても OAuth の範囲外である．

(3) AuthorizationServer は Client の情報や Client が要求している権限の範囲について

表示する．その後，EndUser により権限委譲の許可もしくは拒否の選択が行われる．

(4) AuthorizationServer は，事前に登録された redirect_url もしくは(2)においてオプションで指定された URL に EndUser をリダイレクトさせる．リダイレクト先 URL にパラメータとして EndUser の権限委譲の可否結果を付加する．許可の場合は，その都度生成されるランダムな文字列である AuthorizationCode を付加する．

(5) EndUser は AuthorizationServer からのリダイレクトにより Client のページにアクセスする．このリダイレクトを介して AuthorizationCode が Client に通知される．

(6) Client は，取得した AuthorizationCode，client_id と client_secret を付加して AuthorizationServer に HTTP リクエストを送信する．

(7) AuthorizationServer は Client から送信された client_id，client_secret と AuthorizationCode を検証し問題がなければ，アクセストークンと実際にアクセストークンに設定された scope，expires を HTTP レスポンスのボディに付加して Client に通知する．

以上のシーケンスで EndUser から Client への権限委譲は完了し，以降，Client はアクセストークンを用いて ResourceServer が保持する EndUser の保護されたリソースにアクセスする．

3 XHR2

３．１ XHR 概要

XHR（XMLHttpRequest）とは JavaScript の HTTP 通信機能を提供するオブジェクトの事，もしくは XHR オブジェクトを介して行われる HTTP 通信の事である（以降 XHR と記述した場合は JavaScript による HTTP 通信の事とする）．

XHR オブジェクトによる HTTP リクエストはページの遷移が発生しない．XHR オブジェクトが提供するメソッドを通して，HTTP リクエストの送信先 URL，使用する HTTP メソッドの指定，一部の HTTP リクエストヘッダの設定が可能である．同様に XHR オブジェクトのメソッドを介して HTTP リクエストに対する HTTP レスポンスのデータを取得することが可能である．XHR オブジェクトにより生成される HTTP リクエストには，ブラウザによって自動的に User-Agent ヘッダや Cookie ヘッダが付加される．

３．２ クロスドメイン通信と XHR2

XHR オブジェクトを利用した HTTP 通信は，セキュリティ上の理由より Same Origin Policy[4]にもとづいた制約が課せられる．JavaScript をロードしたドメインと同一のドメインにしか HTTP リクエストを送信することができない．

それに対して，XHR2 では通信先のサーバが明示的にクロスドメインでの HTTP 通信を許可している場合，通信を行うことが可能である．その通信の概要を図 2 の番号と対応させて説明する．

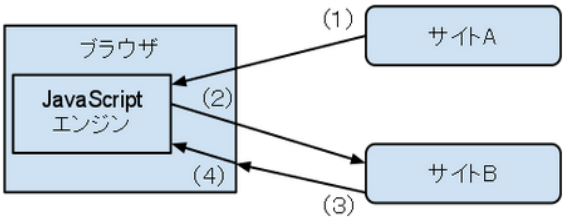

図 2. XHR2 のクロスドメイン通信

(1) サイト A より JavaScript のコードをロードする．

(2) ロードされた JavaScript コードは JavaScript エンジンによって解釈され実行される．XHR オブジェクトを用いてサイト B に HTTP リクエストを送信する際，ブラウザは HTTP リクエストの Origin ヘッダにサイト A のドメイン名をセットし送信する．

(3) サイト B は HTTP リクエストに対する HTTP レスポンスを返す．

(4) ブラウザは，HTTP レスポンスの Access-Control-Allow-Origin ヘッダ[5]に，サイト A

のドメイン名が指定されていた場合のみ，HTTP レスポンスのデータを JavaScript エンジンにわたす．それ以外の場合，JavaScript エンジンはエラーを通知する．

３．３　XHR2 と HTTP 認証

XHRオブジェクトを用いたクロスドメインでのHTTP通信の際，Cookieは付加されない．また，JavaScriptはHTTPレスポンスのSet-Cookieヘッダにアクセス出来ない．そのため，クロスドメイン通信であり，かつ，ユーザを識別する必要がある場合は，XHRオブジェクトのメソッドを介し，パラメータとしてユーザ情報をHTTPリクエストに付加する方法がある．

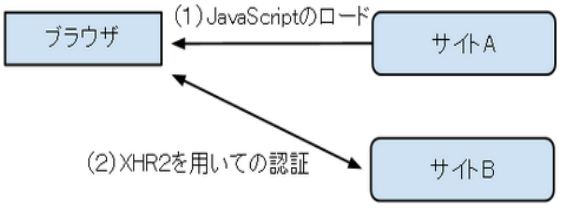

図 3.　XHR2 と認証

しかし，図 3 のようにクロスドメイン通信で，クロスドメイン先のサイトBにおいて認証を行いたい場合，ユーザのクレデンシャル情報をパラメータとして XHR オブジェクトを介した HTTP リクエストに付加する事になり，サイト A の JavaScript にユーザのクレデンシャル情報が渡る事になる．その結果として，サイト A にユーザのクレデンシャル情報を取得される可能性がある．

そこで，ブラウザの HTTP 認証機能を用いることで，サイト A の JavaScript にユーザのクレデンシャル情報を渡さずに，HTTP リクエストにユーザのクレデンシャル情報を付加する．

XHR オブジェクトを介して送信された HTTP リクエストに対して，ブラウザがサイト B から HTTP 認証を要求する Unauthorized レスポンスを受信すると，ブラウザは ID,パスワードの入力画面を表示する．ブラウザはその画面で入力されたクレデンシャル情報を認証方式に合わせて Authorization ヘッダに付加し HTTP リクエストを送り直す．この処理は JavaScript にイベントとして通知されず，入力内容についても JavaScript はアクセス出来ない．HTTP リクエストに対する HTTP レスポンスのみが XHR オブジェクトに渡される．そのため，ユーザのクレデンシャル情報をサイトAからロードした JavaScript に渡すこと無くサイト B でユーザの特定が行える．ただし，XHR オブジェクトを介した HTTP 通信において HTTP 認証を行う際は，XHR オブジェクトの withCredentials プロパティに true が設定されていること及び，サーバ側としては Unauthorized レスポンスの Access-Control-Allow-Credentials ヘッダが true である必要がある．

4　提案システム

４．１　概要

本論文では 3．3 で説明したクロスドメインでの認証手法を用いて，EndUser のクレデンシャル情報を Client に取得されないように，かつ，ページ遷移を伴わない OAuth2.0 の実装を行った．また，XHR2 及び Basic 認証を用いるに当たり，ブラウザの機能より ID とパスワードを打ち込むことを EndUser が Client に対して権限委譲を許可したという意味とすることとした．

４．２　構成

４．２．１　　AuthorizationServer

AuthorizationServer は以下のソフトウェアを用いて構成した．

- Apache Version 2.2.3
- PHP Version 5.3.3
- oauth2-php revision 21

OAuth2.0 ライブラリである oauth2-php[6]がある．

oauth2-php の改修点は主に以下の３点である．

- XHR2 でクロスドメイン通信を行う上で，サーバ側で必要な機能を実装した．EndUser からクロスドメインで送信される HTTP リクエスの Origin ヘッダに格納されているドメイン名より Client を特定する．HTTP レスポンスの Access-Control-Allow-Or

igin ヘッダに Client のドメイン名を設定し，Access-Control-Allow-Credentials ヘッダに true を設定した．

- Basic 認証の機能を PHP で実装した．WWW-Authentication ヘッダに，用いる HTTP 認証の種類を付加し Unauthorized レスポンスを返すようにした．また，それに対する HTTP レスポンスの Authorization ヘッダには EndUser のクレデンシャル情報が格納されている，それを元にユーザの特定及び認証を行うようにした．
- OAuth2.0 で XHR2 を用いるにあたり，OAuth のメッセージの改修を行った．EndUser に対して返信される HTTP レスポンスのボディに EndUser の権限委譲の可否結果及び AuthenticationCode を付加するようにした．

４．２．２ Client

Client は以下のソフトウェアを用いて構成した．

- Apache version 2.2.3

draft-10 の仕様にあわせて OAuth2.0 のメッセージを送信可能な簡易的な JavaScript コードを記述した．

EndUser のブラウザによってその JavaScript コードが実行されると，XHR オブジェクトを生成し，withCredentials プロパティを true に設定する．さらに，AuthorizationServer の認可用ページに対して，client_id や scope を付加した HTTP リクエストを送信する．HTTP リクエストに対する HTTP レスポンスに格納されている AuthorizationCode を取得し，Client に HTTP リクエストを介して提出する．

４．２．３ EndUser

ブラウザは Google Chrome 12.0.742.122 を使用した．

４．３ 動作

提案システムの動作を図4に示す．図1からの変更した部分については破線で示す．

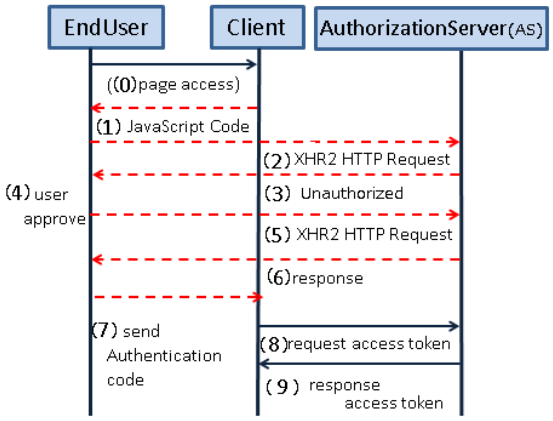

図 4. 提案システムのシーケンス

(1) ClientはEndUserにJavaScriptのコードをロード及び実行させるようなHTMLファイルを送信する．

(2) EndUserのブラウザでJavaScriptが実行され，XHRオブジェクトを介してAuthorizationServerにPOSTメソッドのHTTPリクエストを送信する．この際，HTTPリクエストにclient_idとscope等を付加する．

(3) AuthorizationServerは受け取ったHTTPリクエストのOriginヘッダに格納されている情報よりClientを特定する．その後，XHRオブジェクトのHTTPリクエストに対するHTTPレスポンスとしてUnauthorizedレスポンスを返す．この際，WWW-Authenticationヘッダに認証方式としてBasic認証を設定し，認証要求メッセージとして権限を要求しているClientとその権限の範囲を設定する．

(4) AuthorizationServerからUnauthorizedレスポンスが返されると，EndUserのブラウザは認証要求メッセージとIDとパスワードの入力画面を表示する．EndUserがIDとパスワードを入力することで，Clientへの権限委譲を認可したものと扱う．

(5) EndUserのブラウザはID・パスワードの入力が行われると，(2)で行われたHTTPリクエストの再送を行う．この際，ブラウザがID

及びパスワードをHTTPリクエストのAuthorizationヘッダに付加にする．この処理はJavaScriptには感知されない.

(6) AuthorizationServerは受け取ったHTTPリクエストのAuthorizationヘッダよりEndUserのIDとパスワードを取り出し検証を行う．問題がなければ，AuthorizationCodeを生成しHTTPレスポンスのボディに含め送信する．問題があれば，エラーをHTTPレスポンスのボディに含めて送信する.

(7) AuthorizationServerからHTTPレスポンスを受け取り，そこにAuthorizationCodeが含まれていれば， XHRオブジェクトを介してClientにHTTPリクエストを送信することで，AuthorizationCodeを提出する.

４．４　考察

オーバーヘッドは削減されたが，従来のOAuth2.0 と比較するとデメリットも存在する．従来 AuthorizationServer の認可ページには委譲する権限の範囲を狭めるなどといった機能を持たせることが出来たが，ブラウザの ID・パスワード入力画面では認証要求に際してのメッセージを表示する事しかできない.

また，HTTP認証においてブラウザが表示するID及びパスワードの入力画面で入力したID及びパスワードはブラウザにより記憶される．同一ドメインへのHTTPリクエストには自動でAuthorizationヘッダにクレデンシャル情報が付加されてしまう．これは複数のClientで提案システムを使用する際問題がある．どれか一つのClientでAuthorizationServerとHTTP認証を行うと，他のClientでAuthorizationServerとHTTP認証を行う際，ブラウザが自動でクレデンシャル情報を付加してしまう． HTTP認証を権限委譲の許可として意味づけているため，一度HTTP認証を行うと他のClientにも権限委譲の許可が自動的に行われてしまうことになる．この問題については，Authorizationヘッダが付加されたHTTPリクエストに対して，Unauthorizedレスポンスを返すことによって認証をやり直す方法や，XHRオブジェクトを介して送信されるHTTPリクエストに付加されるOriginヘッダを利用し複数のClientで提案システムを使用していてもそれぞれ区別可能にする方法が考えられるが，今後の課題とする.

5　まとめ

本論文では，XHR2 及び Basic 認証を用いてページ遷移を伴わない OAuth2.0 の実装を行った．XHR2 及び HTTP 認証を用いる事によって，AuthorizationServer の認可用ページ及び認可後に生じる Client のページのロード，描画処理，及びクライアントサイドプアプリケーションの再実行を削除することができた．また，リダイレクトを行わないため Client のページに含まれるクライアントサイドアプリケーションは中断されることはなくアプリケーションの実行状態を管理する必要がなくなった.

6　参考文献

[1] The OAuth 2.0 Protocol draft-10
http://tools.ietf.org/html/draft-ietf-oauth-v2-10

[2] XMLHttpRequest Level 2
http://www.w3.org/TR/XMLHttpRequest2

[3] HTTP Authentication
http://www.ietf.org/rfc/rfc2617.txt

[4] Same origin policy for JavaScript
http://developer.mozilla.org/en/Same_origin_policy_for_JavaScript

[5] Cross-Origin Resource Sharing
http://www.w3.org/TR/cors/

[6] oauth2-php
http://code.google.com/p/oauth2-php/