OAuth2.0 に SPKI 権限証明書を適用した権限委譲方式の提案と実装

齋藤 孝道† 大丸 雅人‡ 西倉 裕太‡ 磯 侑斗‡ 渡邉 貴文‡

† 明治大学, ‡ 明治大学大学院
214-8571 神奈川県川崎市多摩区東三田 1-1-1
saito@cs.meiji.ac.jp
{ce36010, ce46025, ce36004, ce46038}@meiji.ac.jp

あらまし 近年, Web API の利用において, サーバ上で保護されたエンドユーザのリソースの権限委譲を, 別の Web アプリケーションサーバに対して行うケースが増えている. このような権限委譲を実現する手段として, OAuth が利用されている. OAuth により, エンドユーザの ID・パスワードを, Web アプリケーションサーバに渡さずに権限委譲を行うことができる. しかし, RFC6819 では, OAuth におけるセキュリティ上の検討事項が多く挙げられている.

そこで本論文では, RFC6819 で指摘されている, 認可コード及びアクセストークンに関する脅威への対策の一つとして, OAuth2.0 に SPKI 権限証明書を適用した権限委譲方式を提案する.

Delegation Mechanism of Access Authority Extended OAuth2.0 with SPKI Certificate

Takamichi SAITO† Masato OMARU‡ Yuta NISHIKURA‡ Yuto ISO‡ Takafumi WATANABE‡

† Meiji University, ‡ Graduate School of Meiji University
1-1-1, Higashimita, Tama-ku Kawasak-shi, Kanagawa, 214-8571, Japan
saito@cs.meiji.ac.jp, {ce36010, ce46025, ce36004, ce46038}@meiji.ac.jp

Abstract Recently, it is getting popular that, with using Web API, Web Application Server delegates its user's resource authority to other site as its service. OAuth is most promising technology for delegation of authority in Web Application Server. However, since the implementation of OAuth is not easy for everyone, security problems about OAuth in RFC6819 is often reported. Therefore, in this paper, as yet another technology of delegation of authority, we propose a delegation mechanism with applying the SPKI Certificate to OAuth2.0.

1 はじめに

インターネットにおける Web2.0 技術の登場前は, 一つのコンテンツは一つのドメインから提供されることが一般的であった. その後, Web API を提供するサイトや Ajax (Asynchronous JavaScript + XML)といった技術が登場し, 複数の Web サービスを組み合

わせることで新たなサービスを構築するマッシュアップが可能になった.

マッシュアップサービスにおいては，様々なWeb アプリケーションの実現形態が想定できるが，マッシュアップを提供するサーバ上のエンドユーザの保護リソースに対し，アクセス権限を必要とする場合を想定する. マッシュアップサービスを提供する Web アプリケーションサーバ（以降，単に，マッシュアップサービスという）がエンドユーザに変わってアクセス権限を利用する場合，従来ではマッシュアップサービス側で，エンドユーザから ID・パスワードを預かり，保持する運用形態が多かった. このような運用形態の問題点として，マッシュアップサービスはエンドユーザの ID・パスワードを悪用することで，エンドユーザになりすますことが可能になることや，保護リソースの漏洩という問題がある. これらの問題を解決するため，エンドユーザが ID・パスワードをマッシュアップサービスに渡さずに，エンドユーザの保護リソースにアクセスする権限を委譲する仕組みとして OAuth [1]が提案された. しかし，RFC6819 [2]では，OAuth における脅威モデルやセキュリティ上の検討事項が多く挙げられているとことからも推測できる通り，セキュリティの観点で，その構築・運用は容易であるとは言えない. 実際，最近では，普及して久しいにもかかわらず脆弱性といえる問題が指摘された[3].

そこで本論文では，RFC6819 の 4 節において脅威の指摘されている認可コード（後述）及びアクセストークン(後述)に関する脅威への対策の一つとして，[4]を基に，SPKI（Simple Public Key Infrastructure）[5] [6]で利用される SPKI 権限証明書を OAuth2.0 に適用した，別アプローチのアクセス権限委譲方式の提案と実装を行い，その実現可能性と安全性を示す.

2 OAuth 2.0

ここでは OAuth2.0 について説明する.

2.1 概要

OAuth2.0 は，Web アプリケーションサーバに対して，エンドユーザの保護リソースへのアクセス権限の委譲を可能にするフレームワークである.

OAuth は，2006 年末に Twitter の Blaine Cook 氏が，Web API のアクセス権限委譲の仕組みを作ろうとしたことがきっかけで開発が始まった. 2007 年 12 月に OAuth1.0 の仕様が確定され，2012 年 10 月には OAuth2.0 が RFC6749 として登録された.

2.2 OAuth2.0 の用語

・認可コード

エンドユーザ（後述）がアクセス権限の委譲をする際，Authorization Server（後述）から Client（後述）へ発行される文字列. Client は，認可コードを Authorization Server に提示することでアクセストークンを取得できる.

・アクセストークン

Client が，認可コードを Authorization Server に提示した際に発行される文字列. Client は，アクセストークンを Resource Server（後述）に提示することで，Resource Server で管理されているエンドユーザの保護リソースを取得できる.

・エンドユーザ（Resource Owner）

Resource Server 上に保護リソースを持つ主体. また，エンドユーザは保護リソースに対するアクセス権限を持ち，Client にそのリソースへのアクセス権限を委譲する.

・Authorization Server

エンドユーザの認証や，認可コード及びアクセストークンの発行を行うサーバ.

・Resource Server

エンドユーザの保護リソースを管理し，アクセストークンが提示された際にこれを提供するサーバ.

・Client

エンドユーザから権限の委譲を受け，エンドユーザの代理として保護されたリソースへアク

セスを行うアプリケーション．本論文で想定する Client は， Confidential Client である．Confidential Client とは，Resource Server が Client を認証するために Client に発行するパスワードの機密性を保持できる Client である．例えば，Web アプリケーションなどが該当する．また，他に Public Client があるが，安全性の観点から，本論文では議論に含めない．

2.3 動作フロー

OAuth2.0 では，Authorization Code，Implicit，Resource Owner Password Credentials 及び，Client Credentials というフローが用意されている．ここでは本論文で利用する Authorization Code について説明する（図1）．ただし，Client はフローを開始する前に，自らの情報（redirect_uri 等）を Server に登録しておくものとする．redirect_uri とは，Client が認可レスポンス（後述）を受信する際の URI である．

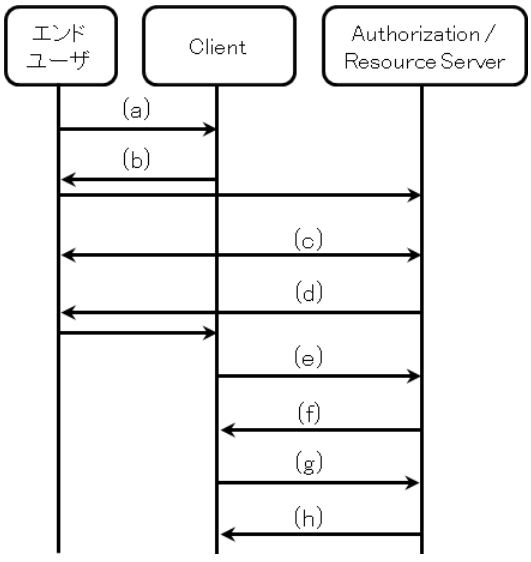

図 1. Authorization Code

(a) エンドユーザは，Client が提供するサービスを受けるため，Client へアクセスする．

(b) Client は， Authorization Server へ認可リクエストを送信する．認可リクエストの送信は，エンドユーザをリダイレクトすることで実現される．認可リクエストには，事前登録した redirect_uri を含める．この場合，redirect_uri は，Client 上の URI である．また，認可リクエストにはに scope という値を含める．scope とは，Client が取得を望むアクセス権限の範囲を示す文字列である．この文字列は，Authorization Server が定義しておく．例えば Authorization Server は，”Client から提示される scope の値が name である場合，エンドユーザの氏名を取得できる Web API へのアクセス権限を要求しているとみなす”という定義を行う．

(c) Authorization Server はエンドユーザを認証する．認証自体は，OAuth2.0 の仕様外である．その後，Authorization Server は，エンドユーザに対して，Client へアクセス権限を委譲してよいかの可否を訪ねる．この際，Authorization Server は，(b)で Client から受信した scope の値を基に，Client が取得を望んでいるアクセス権限をエンドユーザに告知できる．

(d) Authorization Server は Client へ認可レスポンスを送信する．認可レスポンスの送信は，エンドユーザを Client へリダイレクトすることで実現される．その際，(c)でエンドユーザがアクセス権限委譲を許可した場合，Authorization Server は認可コードを生成し，認可レスポンスに含める．送信先の URI は，(b)で指定された redirect_uri である．

(e) Client は独自のサービスを提供するために，取得した認可コードを Authorization Server に対して送信する．その際，(b)で指定した redirect_uri も送信する．

(f) Resource Server は，(e)で取得した redirect_uri が，事前に登録されたものと一致することを確認する．その後，Resource Server は認可コードに対応したアクセストークンを発行し，Client へ送信する．

(g) Client は，(f)で取得したアクセストークンを Resource Server に対して送信する.
(h) Resource Server はアクセストークンの正当性を確認し，それに基づいた保護リソースを提供する.

2.4 脅威モデル

ここでは，RFC6819の4.1.5 節[7] において指摘されている， OAuth2.0における脅威モデルの一つである，Client上のオープンリダイレクタの脅威（Open Redirectors on Client）について説明する.

前提条件として，Clientにオープンリダイレクタ脆弱性が存在し，かつ，Authorization ServerがClientにアクセストークンを発行する際の，redirect_uriの検証（図1のf）に問題がある場合を考える. オープンリダイレクタとは，パラメータで指定された任意のURIにリダイレクトを行うWebページのことである. 例えば，クエリパラメータとして，next=*URI* を指定すると，*URI* へリダイレクトが行われるようなWebページである. また，redirect_uriの検証の問題とは，Authorization Serverが，redirect_uriの完全一致の確認を行わないという問題である. 本来ならば，Authorization Serverは事前登録されたredirect_uriと，Clientがアクセストークンを取得する際に提示されるredirect_uri（図1のe)がパス名を含めて完全に一致することを確認する必要がある. しかし，ドメイン名が一致していることは確認するが，パス名に関しては確認しない場合に問題となる. この脆弱性を悪用されると，攻撃者は認可コード及びアクセストークンを不正に取得し，正規のエンドユーザの保護リソースを不正に取得できてしまう.

オープンリダイレクタに関する脅威が顕在化する場合のフローについて説明する（図2）.

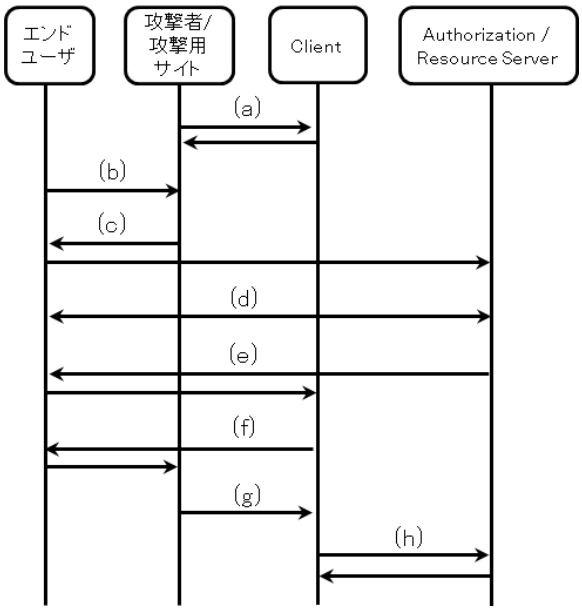

図2. オープンリダイレクタに対する攻撃

(a) 攻撃者は，実際の攻撃に先だって，攻撃対象の Client へアクセスし，Authorization Code のフローを開始する. 攻撃者は，Client が Authorization Server へ送信しようとする認可リクエストをリダイレクト中に止め，認可リクエストに含まれる redirect_uri の改変を行う. 具体的には，認可レスポンスがオープンリダイレクタ経由で攻撃者の用意した Web サイトに送信されるように改変する.
(b) 攻撃者は，エンドユーザをなんらかの方法で攻撃用サイトへ誘導する.
(c) 攻撃者は，(a)で改変した認可リクエストを，なんらかの方法でエンドユーザに送信させる.
(d) Authorization Server はエンドユーザを認証し，エンドユーザは Authorization Server に，アクセス権限の委譲の許可を与える.
(e) Authorization Server は，認可コードを含む認可レスポンスを Client へ送信する. ここでの送信先 URI は，(a)で攻撃者が改変した redirect_uri となることに注意する.
(f) Client は，Authorization Server から認可レスポンスを受け取る. その後，オープン

リダイレクタの機能により，攻撃用サイトへリダイレクトされる．この際，攻撃者はクエリストリングなどを通して，認可レスポンスに含まれる認可コードを取得できる．

(g) 攻撃者は，(f)で取得した認可コードを攻撃対象の Client へ送信する．

(h) Client は，Resource Server へ認可コードを提示してアクセストークンを取得する．このアクセストークンを Resource Server に提示すると，エンドユーザの保護リソースを取得できるため，攻撃者は Client を通してエンドユーザの保護リソースを不正に取得できてしまう．

3 SPKI 権限証明書

本論文では SPKI において用いられている SPKI 権限証明書を利用する．

SPKI 権限証明書は，RFC2962 で策定された権限管理のための証明書である．証明書自体は，X.509v3 証明書[8]と違い，公開鍵とアクセス権限の対応に，権限証明書の発行者がディジタル署名を付与したものである．[9]にも普及する上での課題が示されているが，具体的なサービスの実装が少ないことも，その理由としてある．

本論文で利用する，具体的な SPKI 権限証明書の様式は次のようになっている [10]：

$$<I, S, D, A, V>S(I)$$

・I（*Issuer*）

権限証明書の発行者の公開鍵，もしくは，その主体自体を示す文字列．本論文では，発行者の公開鍵とする．

・S（*Subject*）

権限証明書の発行を受ける主体の公開鍵，もしくは，その主体自体を示す文字列．本論文では，権限を行使する主体の公開鍵とする．

・D（*Delegation*）

ブール値．True，または，False．*Subject* が更に権限を委譲することが可能かどうかを示している．

・A（*Authorization*）

権限を示す．

・V（*Validity*）

証明書の有効期限を示す．

・$S(I)$

SPKI 権限証明書の発行者 I の秘密鍵で，I, S, D, A, V に対してディジタル署名した値．

4 提案手法

ここでは，本論文で提案する権限委譲方式について説明する．

4.1 概要

提案方式は，認可コード及びアクセストークンを利用せず，代わりにエンドユーザの保護リソースに対するアクセス権限を，SPKI 権限証明書によって管理する方式である．

4.2 提案方式の構成

・エンドユーザ

Server（後述）に保持された，自身の保護リソースへのアクセス権限を持ち，Client（後述）にそのリソースへのアクセス権限を委譲する人．

・Server

エンドユーザの保護リソースを管理するサーバ．Client から提示される SPKI 権限証明書に応じて，Client にエンドユーザの保護リソースを提供する．

・Client

エンドユーザの許可のもと，エンドユーザの代理として保護リソースへのアクセスを行う Web アプリケーション．

・Cert1

Server がエンドユーザに対して発行する SPKI 権限証明書．これは，エンドユーザが，”Server 上の自身の保護リソースへのアクセス権限及びその権限を委譲する権限を有すること” を示すものである．Cert1 とする SPKI

権限証明書の具体的な値はそれぞれ以下となる:

Issuer : Server の公開鍵.
Subject : エンドユーザの公開鍵.
Delegation :True
Authorization :Server で管理されるエンドユーザの全ての保護リソースを示す値.
Validity : 任意.
S(I) : Server の公開鍵に対応した秘密鍵で, *Issuer* から *Validity* の全てにディジタル署名した値.

・Cert2

エンドユーザが Client に発行する SPKI 権限証明書. これは, Client が, "Server 上のエンドユーザの保護リソースへのアクセス権限を, エンドユーザから委譲されたこと" を示すものである. Cert2 とする SPKI 権限証明書の具体的な値は以下となる:

Issuer : エンドユーザの公開鍵.
Subject : Client の公開鍵.
Delegation :False
Authorization : Client から提示される scope が入る. scope は, OAuth2.0 のそれで, Server 上のエンドユーザの保護リソースのうち, エンドユーザが Client へアクセス権限を委譲したものを示す値である.
Validity : Client から提示される expires_in が入る. expires_in は, この証明書の有効期限を示す値である.
S(I) : エンドユーザの公開鍵に対応した秘密鍵で, *Issuer*から *Validity*の全てにディジタル署名した値.

4.3 環境

・Server
OS: CentOS6.4
(kernel2.6.32-358.11.1.e16.x86_64)
HTTP Server : Apache HTTP Server 2.2.15
サーバサイドスクリプト: PHP 5.3.3
暗号ライブラリ: OpenSSL 1.0.1e

・Client
OS: CentOS 6.4
(kernel2.6.32-358.11.1.e16.x86_64)
HTTP Server : Apache HTTP Server 2.2.15
サーバサイドスクリプト: PHP 5.3.3
暗号ライブラリ: OpenSSL 1.0.1e

・エンドユーザ
OS: Windows 7 Home Premium
ライブラリ:Desktop PHP [11]
暗号ライブラリ:OpenSSL 1.0.1e

4.4 動作フロー

提案システムの動作フローの説明を行う(図 3). ただし, フローを開始する前に, エンドユーザ及び Client は自身の公開鍵証明書を Server に登録するものとする. また, エンドユーザは Server から Cert1 を取得しておくものとする. ここで, Cert1 の例を以下に示す:

$$<I, S, D, A, V>S(I)$$

Issuer : *P(S)*
Subject : *P(E)*
Delegation :True
Authorization : Profile+Image
Validity : 1410340423
S(I): $<I, S, D, A, V>P^{-1}(S)$

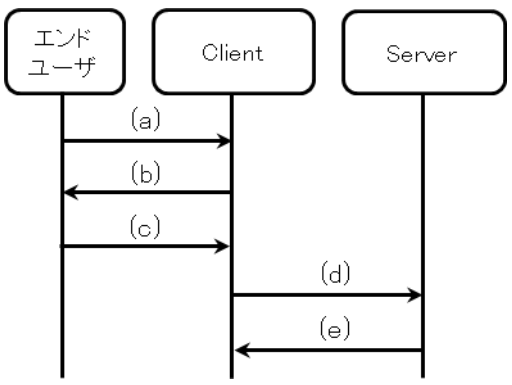

図 3. 提案システムの動作フロー

(a) エンドユーザは Client にアクセスする.
(b) Client は自らの公開鍵証明書(P(C)), scope 及び expires_in をエンドユーザに提示する.
(c) エンドユーザは Client から取得した公開鍵証明書, scope 及び expires_in の値を検証する. 公開鍵証明書の検証では, TLS 及び PKI のフレームワークを利用することで, 正当性を確認する. エンドユーザが Client にアクセス権限の委譲を許可する場合, これら3つの値を基に, Cert2を発行し, Cert1とともに Client へ送信する.
ここで, Cert2 は以下となる:

$$<I, S, D, A, V>S(I)$$

Issuer : *P(E)*
Subject : *P(C)*
Delegation :True
Authorization : Profile
Validity : 1409840157
$S(I)$: $<I, S, D, A, V>P^{-1}(E)$

(d) Client はエンドユーザから取得した Cert1 及び Cert2 を Server へ送信する.
(e) Server は, 予め取得している公開鍵(P(E))Cert1 及び Cert2 のディジタル署名を検証することで, それぞれの正当性を確認する. その後, Cert2 に記載されている *Auhtorization* の値に基づいたエンドユーザの保護リソースを, Cert2 に含まれる Client の公開鍵(P(C))で暗号化し, Client に提供する. Client は, 自身の秘密鍵でこれを復号することで, エンドユーザの保護リソースを取得できる.

4.5 実装の概要

Client 及び Server は, PHP を用いて実装した. 特に, 公開鍵暗号化方式による署名の作成, 検証及び暗号化の処理では, OpenSSL を利用した.

エンドユーザが利用するアプリケーションは, Desktop PHP を利用し, Windows アプリケーションとして実装した. Desktop PHP は, Windows アプリケーションを PHP で実現できるライブラリである. また, 公開鍵暗号化方式による署名の作成及び検証の処理では, OpenSSL を利用した.

4.6 評価

2.4 節で示したように, OAuth2.0 では, 認可コードやアクセストークンが漏洩することにより, 攻撃者がエンドユーザの保護リソースを不正に取得できてしまう. 提案システムは, 認可コード及びアクセストークンを利用せず, SPKI 権限証明書を利用する. また, エンドユーザの保護リソースは SPKI 権限証明書 Cert2 に含まれる Client の公開鍵で暗号化されて提供される. よって, 攻撃者がなんらかの方法で Cert1 及び Cert2 を取得したとしても, Client の秘密鍵を知らない限りは, エンドユーザの保護リソースを復号できず, これを不正に取得することはできない.

5 まとめ

本論文では, RFC6819 の 4 章で指摘されている認可コード及びアクセストークンの漏洩に関する脅威への対策として, SPKI 権限証明書を利用したアクセス権限委譲方式を提案し, 実装した.

参考文献

[1] http://tools.ietf.org/rfc6749
[2] http://tools.ietf.org/rfc6819
[3] Covert Redirect Vulnerability Related to OAuth 2.0 and OpenID, http://tetraph.com/covert_redirect/oauth2_openid_covert_redirect.html
[4] 西倉裕太, 大丸雅人, 今野真希, 磯侑斗, 齋藤孝道, 2014
OAuth2.0 に SPKI を適用したアクセス

権限委譲方法の提案，第 76 回情報処理学会全国大会
[5] http://tools.ietf.org/rfc2962
[6] http://tools.ietf.org/rfc2963
[7] http://tools.ietf.org/html/rfc6819#sections-4.1.5
[8] https://www.ietf.org/rfc/rfc2459
[9] Sebastian Wiesner, Ralph Holz, 2013 Simple PKI, Seminar FI & IITM, WS2012/2013
[10] 齋藤孝道，梅澤健太郎，奥乃博，2000 個人情報の扱いを考慮したアクセス制御の一方法，インターネットコンファレンス 2000
[11] https://code.google.com/p/phpdesktop/