mitrelogo-verysmall-white.png

Secure RESTful Interfaces: Draft Profiles for the Use of OAuth 2.0

Draft Version 1.3

October 15, 2014

The views, opinions and/or findings contained in this report are those of The MITRE Corporation and should not be construed as an official government position, policy, or decision, unless designated by other documentation.

Approved for Public Release; Distribution Unlimited. Case Number 14-3178

Notice: This technical data was produced for the U. S. Government under Contract Numbers VA791-P-0032, VA791-9-0042, VA798A-11-P-0015, and VA118A-13-D-0037 and is subject to Federal Acquisition Regulation Clause 52.227-14, Rights in Data – General, Alt. II (JUN 1987), Alt. III (JUN 1987) and Alt. IV (JUN 1987).

No other use other than that granted to the U. S. Government under that Clause is authorized without the express written permission of The MITRE Corporation.

For further information, please contact The MITRE Corporation, Contracts Office, 7515 Colshire Drive, McLean, VA 22102-7539, (703) 983-6000.

©2014 The MITRE Corporation. This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

Sponsor: Department of Veterans Affairs

Contract No.: VA791-P-0042

Project No.: 40144028

**McLean, VA**

J. Richer

M. Russell

May 2014

This page intentionally left blank.

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Description |
| 1.0 | 6/30/2014 | J. Richer, M. Russell | Initial draft |
| 1.1 | 8/15/2014 | M. Russell | Incorporated additional edits from J. Richer; added token claim examples |
| 1.2 | 9/11/2014 | M. Russell | Reorganized document, moving VA-specific considerations to the introduction to enable the profile to be shared broadly |
| 1.3 | 10/15/2014 | M. Russell | Added notices and rights legend for public release; made audience claim validation optional |

Table of Contents

[Introduction v](#_Toc398281269)

[Draft Secure RESTful Interface Profile for the Use of OAuth 2.0 1](#_Toc398281270)

[1. Terminology 1](#_Toc398281271)

[1. OAuth 2.0 Client Profiles 2](#_Toc398281272)

[1.1. Client Types 2](#_Toc398281273)

[1.1.1. Full Client with User Delegation 3](#_Toc398281274)

[1.1.2. Browser-embedded Client with User Delegation 3](#_Toc398281275)

[1.1.3. Direct Access Client 3](#_Toc398281276)

[1.2. Requests to the Token Endpoint 4](#_Toc398281277)

[1.3. Requests to the Authorization Endpoint 5](#_Toc398281278)

[2. Client Registration 5](#_Toc398281279)

[2.1. Redirect URI 5](#_Toc398281280)

[2.2. Dynamic Registration 5](#_Toc398281281)

[3. OAuth 2.0 Server Profile 6](#_Toc398281282)

[3.1. JWT Bearer Tokens 7](#_Toc398281283)

[3.2. Token Lifetimes 7](#_Toc398281284)

[3.3. Token Revocation and Introspection 8](#_Toc398281285)

[4. Requests to the Protected Resource 8](#_Toc398281286)

[6.1 Client Authentication to the Protected Resource 8](#_Toc398281287)

[5. Scopes 9](#_Toc398281288)

[6. Advanced OAuth Security Options 9](#_Toc398281289)

[6.1. Client TLS Authentication 9](#_Toc398281290)

[6.2. Proof of Possession Tokens 10](#_Toc398281291)

[List of Acronyms 11](#_Toc398281292)

# Introduction

The attached profile for OAuth 2.0 was developed in support of the Secure RESTful Interfaces Profile project for the Department of Veterans Affairs (VA) Deputy Chief Information Officer (CIO), Office of Information and Technology (OIT), Architecture, Strategy, and Design (ASD).

This profile specification is intended for general application across a wide range of VA use cases. It builds on prior OAuth profiles developed by MITRE in support of the RESTful Health Exchange (RHEx) [1] and BlueButton+ REST [2] initiatives. While those past efforts addressed specific health care use cases, this document defines general guidance applicable both within and beyond the health care domain.

It is anticipated that this document will be revised through discussions with VA stakeholders and external organizations including the Office of the National Coordinator for Health IT (ONC) and the Open Source Electronic Health Record Alliance (OSEHRA). In addition, emerging security threat information, such as novel attacks on OAuth or advances in cryptanalysis, may require updates to this OAuth profile.

In developing this OAuth profile, MITRE identified two recommendations for the VA:

* For use cases involving access to Electronic Health Record (EHR) data, an industry standard scheme for defining OAuth scopes would facilitate interoperability among providers, such as provided by the scopes defined in BlueButton+ REST. For this reason, VA should collaborate with its partners in the health IT sector on a common standard. The ONC Standards and Interoperability (S&I) Framework’s Data Segmentation for Privacy initiative [3], is already examining EHR data segmentation, and the products of this effort could provide a basis for defining OAuth scopes for EHR data at useful levels of granularity.
* VA may wish to track the development of emerging standards such as OAuth Proof of Possession tokens, or even participate in the standards definition process. VA participation would help to ensure that the VA’s needs are addressed by the new standards.

This OAuth profile is intended to be shared broadly, and ideally to influence OAuth implementations by other organizations and in other domains besides health care. To this end, the profile is written such that it may be separated from this introduction and stand on its own. The following pages contain the OAuth 2.0 profile.

# Draft Secure RESTful Interface Profile for the Use of OAuth 2.0

This document profiles the OAuth 2.0 web authorization framework for use in the context of securing Representational State Transfer (RESTful) interfaces. The OAuth 2.0 specifications accommodate a wide range of implementations with varying security and usability considerations, across different types of software clients. To achieve this flexibility, the standard makes many security controls optional. OAuth implementations using only the minimum mandatory security measures require minimal effort on the part of developers and users, but they also fail to prevent known attacks and are unsuitable for protecting sensitive data.

The OAuth 2.0 client and server profiles defined in this document serve two purposes:

1. Define a mandatory baseline set of security controls suitable for a wide range of use cases, while maintaining reasonable ease of implementation and functionality
2. Identify optional advanced security controls for sensitive use cases where heightened risks justify more stringent controls that increase the required implementation effort and may reduce or restrict functionality

# Terminology

This profile inherits terminology from RFC6749 [4], RFC6750 [5], RFC2119 [6], and OpenID Connect Core [7].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in [6].

Table 1 below defines some of the terms defined in the standards referenced above and used in this profile specification. The definitions in the table are provided for the reader’s convenience and not meant to supersede or modify the terms’ original authoritative definitions in the respective standards.

Table – Referenced Terms

| Term | Definition |
| --- | --- |
| Access Token | A token issued to the client and passed to the protected resource as proof of access authorization |
| Authorization Code | A token representing the resource owner’s authorization obtained from the authorization endpoint, which can be exchanged for an access token at the token endpoint |
| Authorization Code Grant | An OAuth grant type in which the client obtains an authorization code from the authorization endpoint and exchanges it for an access token at the token endpoint |
| Authorization Endpoint | An authorization server endpoint to which clients submit authorization requests (via resource owner user agent redirection); the authorization endpoint authenticates resource owners and prompts them to approve or deny authorization requests |
| Authorization Server | A server that implements the OAuth 2.0 authorization endpoint, token endpoint, and optionally other OAuth endpoints, and issues authorization tokens to clients. |
| Client | A software application that requests OAuth access tokens from an authorization server |
| Client Credentials Grant | An OAuth grant type used when the client is not acting on behalf of a particular resource owner, in which the client’s credentials are exchanged for an access token at the token endpoint |
| Confidential Client | An OAuth client capable of protecting the confidentiality of its client credentials or otherwise authenticating securely to the authorization server |
| Implicit Grant | An OAuth grant type used for browser-embedded clients, in which an access token is returned to the client directly from the authorization endpoint; the client is not authenticated and the token endpoint is not used |
| Protected Resource | A resource (API, application, or data) to which authorization is granted through the OAuth framework |
| Redirect Uniform Resource Identifier (URI) | A URI submitted by a client as part of an authorization or access token request, indicating the client URI to which the authorization server’s response (including access tokens or authorization codes) should be directed |
| Refresh Token | An optional token issued along with an access token which can be used to obtain a new access token after the initial access token expires |
| Resource Owner | A user with the authority to authorize clients to access protected resources on his/her behalf |
| Resource Owner Password Credentials Grant | An OAuth grant type in which the resource owner’s credentials are provided directly to the client, which exchanges them for an access token at the token endpoint |
| Token Endpoint | An authorization server endpoint to which clients submit authorization codes or client credentials in order to receive access tokens |
| Token Introspection Endpoint | An authorization server endpoint to which protected resources can submit tokens received from clients in order to obtain token metadata such as validity period or approved scope |
| Token Revocation Endpoint | An authorization server endpoint to which clients can submit token revocation requests to indicate that tokens will no longer be used |

# OAuth 2.0 Client Profiles

All clients MUST conform to applicable recommendations found in the Security Considerations sections of RFC6749 [4] and those found in the OAuth 2.0 Threat Model and Security Considerations document [8].

## Client Types

The following profile descriptions give patterns of deployment for use in different types of client applications based on the OAuth grant type. The resource owner password credentials grant type defined in RFC6749 [4] is intentionally omitted from this discussion, and its use is not permitted under these profiles. Additional grant types, such as assertions, chained tokens, or other mechanisms, are out of scope of this profile and must be covered separately by appropriate profile documents.

### Full Client with User Delegation

This client type applies to clients that act on behalf of a particular resource owner and require delegation of that user’s authority to access the protected resource. Furthermore, these clients are capable of interacting with a separate web browser application to facilitate the resource owner’s interaction with the authentication endpoint of the authorization server.

These clients MUST use the authorization code flow of OAuth 2 by sending the resource owner to the authorization endpoint to obtain authorization. The user MUST authenticate to the authorization endpoint. The user’s web browser is then redirected back to a URI hosted by the client, from which the client can obtain an authorization code passed as a query parameter. The client then presents that authorization code along with its own credentials to the authorization server’s token endpoint to obtain an access token.

The authorization code flow is supported only for confidential clients. Examples of this client type include web applications and native applications that can store installation-instance-specific client credentials securely. Client credentials MUST NOT be shared among instances of a given piece of client software.

This client type MAY request and be issued a refresh token if the security parameters of the access request allow for it.

### Browser-embedded Client with User Delegation

This client type applies to clients that act on behalf of a particular resource owner and require delegation of that user’s authority to access the protected resource. Furthermore, these clients are embedded within a web browser and effectively share an active session between systems.

These clients use the implicit flow of OAuth 2 by sending a resource owner to the authorization endpoint to obtain authorization. The user MUST authenticate to the authorization endpoint. The user’s web browser is then redirected back to a URI hosted by the client, from which the client can directly obtain an access token. Since the client itself never authenticates to the server and the token is made available directly to the browser, this flow is appropriate only for clients embedded within a web browser, such as a JavaScript client with no back-end server component. Wherever possible, it is preferable to use the authorization code flow due to its superior security properties.

This client type MUST NOT request or be issued a refresh token. Access tokens issued to this type of client MUST be short lived and SHOULD be tied to the user’s authentication session with the client.

### Direct Access Client

This profile applies to clients that connect directly to protected resources and do not act on behalf of a particular resource owner, such as those clients that facilitate bulk transfers.

These clients use the client credentials flow of OAuth 2 by sending a request to the token endpoint with the client’s credentials and obtaining an access token in the response. Since this profile does not involve an authenticated user, this flow is appropriate only for trusted applications, such as those that would traditionally use a developer key.

For example, a partner system that performs bulk data transfers between two systems would be considered a direct access client.

This client type MUST NOT request or be issued a refresh token.

## Requests to the Token Endpoint

Full clients and direct access clients as defined above MUST authenticate to the authorization server’s token endpoint using a JavaScript Object Notation (JSON) Web Token (JWT) assertion as defined by the JWT Profile for OAuth 2.0 Client Authentication and Authorization Grants [9] and the “private\_key\_jwt” method defined in OpenID Connect Core [7]. The assertion MUST use the claims as follows:

* *iss:* the client ID of the client creating the token
* *sub:* the client ID of the client creating the token
* *aud:* the URL of the authorization server’s token endpoint
* *iat:* the time that the token was created by the client
* *exp:* the expiration time, after which the token MUST be considered invalid
* *jti:* a unique identifier generated by the client for this authentication. This identifier MUST contain at least 128 bits of entropy and MUST NOT be re-used by any subsequent authentication token.

The following sample claim set illustrates the use of the required claims for a client authentication JWT as defined in this profile; additional claims MAY be included in the claim set:

{

"iss":"client1234@example.com",

"aud":"https://authz-svr.example.com/oauth/access\_token ",

"sub":"client1234@example.com",

"iat":1406538000,

"exp":1406538060,

"jti":"hTNObzOO0Dv/SBLL3veKng=="

}

The JWT assertion MUST be signed by the client using the client’s private key. See Section 2 (Client Registration) for mechanisms by which the client can make its public key known to the server. The authorization server MUST support the RS256 signature method (the Rivest, Shamir, and Adleman (RSA) signature algorithm with a 256-bit hash) and MAY use other asymmetric signature methods listed in the JSON Web Algorithms (JWA) specification [10].

Authorization servers MAY require some clients to additionally authenticate using mutual Transport Layer Security (TLS) authentication, with the client’s TLS certificate having been registered at the authorization server. Due to problems inherent in configuring a large mutual TLS network at scale, it is RECOMMENDED by this profile that such authentication be limited to instances where the security benefits sufficiently outweigh the complications.

## Requests to the Authorization Endpoint

Full clients and browser-embedded clients making a request to the authorization endpoint MUST use an unpredictable value for the *state* parameter with at least 128 bits of entropy. Clients MUST validate the value of the *state* parameter upon return to the redirect URI and MUST ensure that the *state* value is securely tied to the user’s current session (e.g., by relating the *state* value to a session identifier issued by the client).

Clients MUST include their full redirect URIs in the authorization request. To prevent open redirection and other injection attacks, the authorization server MUST match the entire redirect URI using a direct string comparison against registered values and MUST reject requests with invalid or missing redirect URIs.

# Client Registration

All clients MUST register with the authorization server. For client software that may be installed on multiple client instances, such as native applications or web application software, each client instance MUST receive a unique client identifier from the authorization server.

Clients using the authorization code or client credentials grant type MUST have a public and private key pair for use in authentication to the token endpoint. These clients MUST register their public keys in their client registration metadata by either sending the public key directly in the *jwks* field or by registering a *jwks\_uri* which MUST be reachable by the authorization server. It is RECOMMENDED that clients us a *jwks\_uri* if possible as this allows for key rotation more easily.

## Redirect URI

Clients using the authorization code or implicit grant types MUST register their full redirect URIs. The Authorization Server MUST validate the redirect URI given by the client at the authorization endpoint using strict string comparison.

A client MUST protect the values passed back to its redirect URI by ensuring that the redirect URI is one of the following:

* Hosted on a website with TLS protection (a Hypertext Transfer Protocol – Secure (HTTPS) URI)
* Hosted on the local domain of the client (e.g., http://localhost/)
* Hosted on a client-specific non-remote-protocol URI scheme (e.g., myapp://)

Clients MUST NOT have URIs in more than one category and SHOULD NOT have multiple redirect URIs on different domains.

Clients MUST NOT forward values passed back to their redirect URIs to other arbitrary or user-provided URIs (a practice known as an “open redirector”).

## Dynamic Registration

Authorization servers MAY support dynamic client registration, and clients MAY register using the draft Dynamic Client Registration Protocol [11] for authorization code or implicit grant types. Clients MUST NOT dynamically register for the client credentials grant type. (It should be noted that dynamic registration is a draft standard, so there is a risk that future substantive changes to the protocol specification could impact implementations. However, at the time of this writing, the draft standard has passed working group last call and is scheduled to move forward to the IESG, making this risk fairly minimal in practice.

Authorization servers MUST signal to end users that a client was dynamically registered on the authorization screen. Authorization servers MAY issue signed software statements as described in [11] to client software developers. The software statement can be used to tie together many instances of the same client software that will be run, dynamically registered, and authorized separately at runtime. The software statement MUST include the following client metadata parameters:

* *redirect\_uris*: array of redirect URIs used by the client; subject to the requirements listed in Section 2.1
* *grant\_types:* grant type used by the client; must be “authorization\_code” or “implicit”
* *jwks\_uri or jwks:* client’s public key in JWK Set format; if *jwks\_uri* is used it MUST be reachable by the Authorization Server and point to the client’s public key set
* *client\_name:* human-readable name of the client
* *client\_uri:* URL of a web page containing further information about the client

# OAuth 2.0 Server Profile

All servers MUST conform to applicable recommendations found in the Security Considerations sections of [4] and those found in the OAuth Threat Model Document [8].

The authorization server MUST support the authorization code, implicit, and client credentials grant types as described above. The authorization server MUST limit each registered client (identified by a client ID) to a single grant type only. The authorization server MUST enforce client authentication as described above for the authorization code and client credentials grant types. The authorization server MUST validate all redirect URIs for authorization code and implicit grant types.

The authorization server MUST protect all communications to and from its OAuth endpoints using TLS. The authorization server MUST provide an OpenID Connect service discovery endpoint listing the components relevant to the OAuth protocol:

* *authorization\_endpoint*: the server’s authorization endpoint
* *token endpoint*: the server’s token endpoint
* *introspection\_endpoint*: the server’s token introspection endpoint
* *revocation\_endpoint*: the server’s token revocation endpoint
* *registration\_endpoint*: the server’s dynamic client registration endpoint (if supported)
* *issuer*: the value of the “iss” field used in all issued tokens

If the authorization server is also an OpenID Connect Provider, it MUST provide a discovery endpoint meeting the requirements listed in Section 6 of this document’s companion OpenID Connect profile [12].

## JWT Bearer Tokens

The server MUST issue tokens as JWTs with, at minimum, the following claims:

* *iss:* the issuer URL of the server that issued the token
* *azp:* The client id of the client to whom this token was issued
* *sub:* The identifier of the end-user that authorized this client, or the client id of a client acting on its own behalf (such as a bulk transfer)
* *kid:* the key ID of the keypair used to sign this token
* *exp:*the expiration time, after which the token MUST be considered invalid
* *jti:* A unique JWT Token ID value with at least 128 bits of entropy. This value MUST not be re-used in another token. Clients MUST check for reuse of *jti* values and reject all tokens issued with duplicate *jti* values.

The server MAY issue tokens with an *aud* (audience) claim whose value is an array containing the identifier(s) of protected resource(s) for which the token is valid, if this information is known. The *aud* claim may contain multiple values if the token is valid for multiple protected resources.

The following sample claim set illustrates the use of the required claims for an access token as defined in this profile; additional claims MAY be included in the claim set:

{

"iss":"https://jwt-idp.example.com",

"azp":"client1234@example.com”,

"sub":"jane.doe@example.com",

"kid":"1e9gdk7",

"exp":1406538000,

"jti":"hTNObzOO0Dv/SBLL3veKng=="

}

The access tokens MUST be signed with JSON Web Signature (JWS). The authorization server MUST support the RS256 signature method for tokens and MAY use other asymmetric signing methods.

Refresh tokens SHOULD be signed with JWS using the same public key and contain the same set of claims as the access tokens.

The authorization server MAY encrypt access tokens and refresh tokens. Access tokens MUST be encrypted using the public key of either the protected resource or the authorization server itself. Refresh tokens MUST be encrypted using the authorization server’s public key.

## Token Lifetimes

This profile provides RECOMMENDED lifetimes for different types of tokens issued to different types of clients. Specific applications MAY issue tokens with different lifetimes. Any active token MAY be revoked at any time.

For clients using the authorization code grant type, access tokens SHOULD have a valid lifetime no greater than one hour, and refresh tokens (if issued) SHOULD have a valid lifetime no greater than twenty-four hours.

For clients using the implicit grant type, access tokens SHOULD have a valid lifetime no greater than fifteen minutes.

For clients using the client credentials grant type, access tokens SHOULD have a valid lifetime no greater than six hours.

## Token Revocation and Introspection

The authorization server MUST supply token revocation [13] and token introspection [14] endpoints to allow clients and protected resources to manage the lifecycle of issued tokens.

Token revocation allows a client to signal to an authorization server that a given token will no longer be used. A client MUST immediately discard the token and not use it again after revoking it.

Token introspection allows a protected resource to query the authorization server for metadata about a token. A protected resource MAY cache the response from the introspection endpoint for a period of time no greater than half the lifetime of the token. A protected resource MUST NOT accept a token that is no longer valid according to the response from the introspection endpoint.

The authorization server MUST require authentication for both the revocation and introspection endpoints as described in the Client Authentication section of this document. Clients calling the revocation endpoint SHOULD use the same credentials as used at the token endpoint. Protected resources calling the introspection endpoint MUST use credentials distinct from any other OAuth client registered at the server.

# Requests to the Protected Resource

The protected resource MUST support bearer tokens passed in the Authentication header as defined by RFC 6750 [5]. Protected resources MAY support the form-parameter or query-parameter methods in RFC 6750. Authorized requests MUST be made over TLS, and clients MUST validate the protected resource server’s certificate.

The protected resource MUST check the audience claim, if it exists in the token, to ensure that it includes the protected resource’s identifier. The protected resource MUST ensure that the rights associated with the token are sufficient to grant access to the resource. One mechanism for doing this is by querying the scopes associated with the token from the authorization server’s token introspection endpoint.

## 6.1 Client Authentication to the Protected Resource

Normally with an OAuth bearer token, the client does not separately authenticate to the protected resource. However, some protected resources in high value environments MAY require the client to authenticate to the protected resource in addition to presenting an access token. With such protected resources, the client MUST authenticate using a JWT assertion signed with its private key as described in Section 1.2. A protected resource MAY allow a client to authenticate using mutual TLS, either in lieu of or in addition to the JWT assertion. In such cases, a protected resource MUST ensure that the identifier of the client is included in the “azp” field (authorized presenter) of the access token.

# Scopes

Scopes define individual pieces of authority that can be requested by clients, granted by resource owners, and enforced by protected resources. Specific scope values will be highly dependent on the specific types of resources being protected in a given interface. OpenID Connect, for example, defines scope values to enable access to different attributes of user profiles.

Protected resources MUST define and document which scopes are required for access to the resource.

Authorization Servers SHOULD define and document default scope values that will be used if an authorization request does not specify a requested set of scopes.

# Advanced OAuth Security Options

The preceding OAuth profiles provide a level of security adequate for a wide range of use cases, while still maintaining relative ease of implementation and usability for developers, system administrators, and end users. The following are some additional security measures that can be employed for use cases where elevated risks justify the use of additional controls at the expense of implementation effort and usability. This section also addresses future security capabilities, currently in the early draft stages, being added to the OAuth standard suite.

## Client TLS Authentication

The OAuth 2.0 specification requires the use of TLS to protect several different connections among the parties involved in an OAuth transaction, but in each case only server authentication is required. Clients could additionally be required to negotiate a mutually-authenticated TLS connection:

* When connecting to the Authorization Server’s Token Endpoint to retrieve access and refresh tokens
* When connecting to protected resources

Stronger client authentication to the Token Endpoint reduces the risk of a captured Authorization Code being used to obtain tokens. Stronger client authentication to the protected resource, combined with validating that the authenticated client is the one to which the token was issued through the *azp* token claim, reduces the risk of captured tokens being used by unauthorized clients. In both cases, mutual TLS authentication provides much stronger protection against man-in-the-middle attacks than server authentication alone.

Apart from the difficulty of implementing Public Key Infrastructure (PKI) solutions in distributed, cross-organization settings, one concern with this approach is the clients’ highly variable capabilities to protect private keys. Web application clients may be able to provide strong protection, but with native clients such as mobile apps, the key may be stored in a hardware security module or in plaintext in flash storage.

## Proof of Possession Tokens

OAuth proof of possession tokens are currently defined in a set of drafts under active development in the Internet Engineering Task Force (IETF) OAuth Working Group. While a bearer token can be used by anyone in possession of the token, a proof of possession token is bound to a particular symmetric or asymmetric key issued to, or already possessed by, the client. The association of the key to the token is also communicated to the protected resource; a variety of mechanisms for doing this are outlined in the draft OAuth 2.0 Proof-of-Possession (PoP) Security Architecture [15]. When the client presents the token to the protected resource, it is also required to demonstrate possession of the corresponding key (e.g., by creating a cryptographic hash or signature of the request). Proof of Possession tokens are somewhat analogous to the Security Assertion Markup Language’s (SAML’s) Holder-of-Key mechanism for binding assertions to user identities.

Proof of possession could prevent a number of attacks on OAuth that entail the interception of access tokens by unauthorized parties. The attacker would need to obtain the legitimate client’s cryptographic key along with the access token to gain access to protected resources. Additionally, portions of the HTTP request could be protected by the same signature used in presentation of the token. Proof of possession tokens may not provide the same strong protections against man-in-the-middle attacks as PKI authentication, but they are far less challenging to implement on a distributed scale.

# List of Acronyms

| Acronym | Definition |
| --- | --- |
| ASD | Architecture, Strategy, and Design |
| CIO | Chief Information Officer |
| EHR | Electronic Health Record |
| HTTPS | HTTP Secure |
| IETF | Internet Engineering Task Force |
| IT | Information Technology |
| JSON | JavaScript Object Notation |
| JWA | JSON Web Algorithms |
| JWS | JSON Web Signature |
| JWT | JSON Web Token |
| OIT | Office of Information & Technology |
| ONC | Office of the National Coordinator for Health IT |
| OSEHRA | Open Source Electronic Health Record Alliance |
| PKI | Public Key Infrastructure |
| REST | Representational State Transfer |
| RHEx | RESTful Health Exchange |
| RSA | Rivest, Shamir, and Adleman |
| S&I | Standards & Interoperability |
| SAML | Security Assertion Markup Language |
| TLS | Transport-Layer Security |
| URI | Uniform Resource Identifier |
| VA | Veterans Affairs |

References

|  |  |
| --- | --- |
| [1] | S&I Framework, "RHEx - Powering Secure, Web Base Health Data Exchange," [Online]. Available: http://wiki.siframework.org/RHEx. [Accessed June 2014]. |
| [2] | S&I Framework, "BlueButton Plus Initiative," [Online]. Available: http://wiki.siframework.org/BlueButton+Plus+Initiative. [Accessed 2014 June]. |
| [3] | S&I Framework, "Data Segmentation for Privacy Homepage," [Online]. Available: http://wiki.siframework.org/Data+Segmentation+for+Privacy+Homepage. [Accessed June 2014]. |
| [4] | D. Hardt, "The OAuth 2.0 Authorization Framework," October 2012. [Online]. Available: http://tools.ietf.org/html/rfc6749. [Accessed June 2014]. |
| [5] | M. Jones and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage," October 2012. [Online]. Available: http://tools.ietf.org/html/rfc6750. [Accessed June 2014]. |
| [6] | S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," March 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2119.txt. [Accessed June 2014]. |
| [7] | N. e. a. Sakimura, "OpenID Connect Core 1.0," 25 February 2014. [Online]. Available: http://openid.net/specs/openid-connect-core-1\_0.html. [Accessed May 2014]. |
| [8] | T. Lodderstedt, M. McGloin and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations," January 2013. [Online]. Available: http://tools.ietf.org/html/rfc6819#page-46. [Accessed May 2014]. |
| [9] | M. Jones, B. Campbell and C. Mortimore, "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants," 28 April 2014. [Online]. Available: http://tools.ietf.org/html/draft-ietf-oauth-jwt-bearer-09. [Accessed June 2014]. |
| [10] | M. Jones, "JSON Web Algorithms (JWA)," 20 June 2014. [Online]. Available: http://www.ietf.org/id/draft-ietf-jose-json-web-algorithms-29.txt. [Accessed 21 June 2014]. |
| [11] | J. Richer, M. Jones, J. Bradley, M. Machulak and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol," 22 May 2014. [Online]. Available: http://tools.ietf.org/html/draft-ietf-oauth-dyn-reg-17. [Accessed June 2014]. |
| [12] | The MITRE Corporation, "Secure RESTful Interfaces: Draft Profiles for the Use of OpenID Connect: Draft Version 1.0," June 30, 2014. |
| [13] | T. Lodderstedt, S. Dronia and M. Scurtescu, "OAuth 2.0 Token Revocation," August 2013. [Online]. Available: http://tools.ietf.org/html/rfc7009. [Accessed June 2014]. |
| [14] | J. Richer, "OAuth Token Introspection," 1 May 2013. [Online]. Available: http://tools.ietf.org/html/draft-richer-oauth-introspection-04. [Accessed May 2014]. |
| [15] | P. Hunt, J. Richer, W. Mills, P. Mishra and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture," 3 April 2014. [Online]. Available: http://tools.ietf.org/html/draft-hunt-oauth-pop-architecture-00#section-1. [Accessed June 2014]. |