Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2011

N. Neumann, Ed.
F. Tegeler
X. Fu
University of Goettingen

October 18, 2010

Secure Token Transfer Protocol (STTP)
draft-neumann-oauth-token-transfer-00

Abstract

The Secure Token Transfer Protocol (STTP) provides the means for two entities to exchange a set of tokens that is needed to perform a certain task such as authentication. The exact context of the tokens and their further usage is outside the scope of the protocol. STTP is intended to be employed to in case a mechanism to securely transfer tokens is missing for a particular scenario or context within other protocols.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 21, 2011.

Copyright Notice

Table of Contents

1. Overview

Tokens are commonly used in access authentication and authorization, for example, in a rather generic way in form of usernames and passwords or in more complex ways such as OAuth access tokens. The Secure Token Transfer Protocol (STTP) provides means to securely transfer such access tokens between different entities that might be involved in the provisioning of such tokens. While tokens are commonly required for authentication and authorization purposes within protocols, they do not always provide means to securely transport or provision them in every scenario. If such means are missing for a specific protocol or scenario, STTP can be used for such tasks.

Figure 1 shows an example where STTP is used to transport a set of tokens between three entities to access a resource that is available on a fourth entity. Client A wants to access a protected resource on a server. Between Client A and the server an application specific authentication or authorization protocol is employed (e.g. HTTP Basic Access Authentication or OAuth) within an application specific workflow to access a protected resource (A). The server in turn requests an authentication or authorization protocol specific set of tokens in order to grant the Client access to the requested resource (B). As a set of appropriate tokens is not available on Client A it uses STTP to request a set of tokens from Client B which is under the control and trusted by the resource owner (C). Client A and Client B can be running in different applications and even on different devices. If the user approves the token request, Client B will again use STTP to request a set of tokens that matches the request from

Client A from the appropriate authentication or authorization server (D). The corresponding request can be authenticated within STTP if the server requires so. If the user is authorized to access the protected resource the authentication or authorization server will issue a corresponding set of tokens that are opaque to STTP (E). Client B transfers this set of tokens to Client A (F) which in turn can use it in a protocol specific way to authenticate or authorize the access request to the protected resource (G).

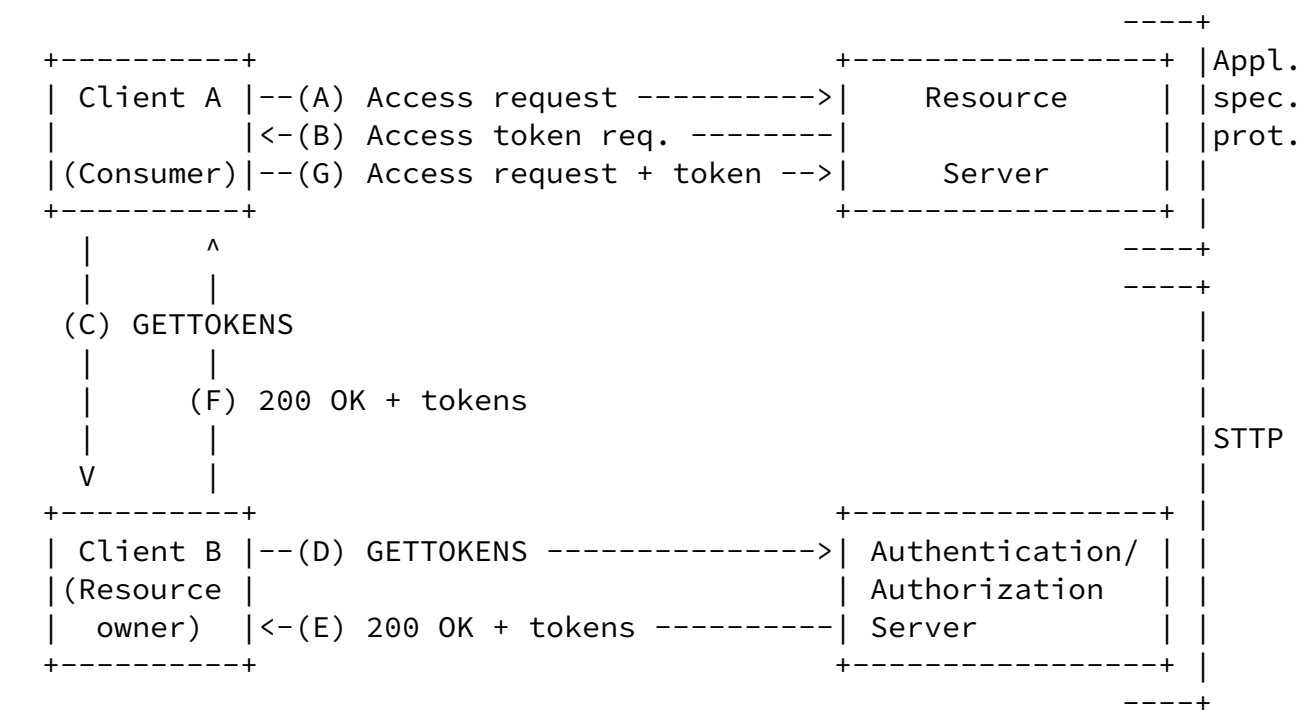

Figure 1: STTP Example

1.1. Requirements Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Terminology

The following terminology is used throughout this document:

Token A data string that is required by another protocol to perform a specific task such as authentication or authorization. The exact context and the further use of tokens is outside the scope of this document. Tokens are opaque to STTP.

Client The entity requesting a set of tokens.

Server The entity that a set of tokens is requested from.

2. Usage Scenarios

This document addresses a number of particular scenarios where STTP could be used that are detailed below.

2.1. OAuth

The OAuth 2.0 Protocol [I-D.ietf-oauth-v2] specifies a workflow that requires end-user authorization. To acquire end-user authorization an application needs to either access a HTTP resource (the end-user authorization endpoint) with an appropriate user-agent or have access to the user's credentials and perform HTTP Basic authentication. In cases where an application does not have access to such an appropriate user-agent or where HTTP Basic authentication is not supported, STTP can be used to request the corresponding OAuth tokens.

OAuth also requires the transfer of an Access Grant and an Access Token between the entity accessing the resource server and the application accessing the authorization server. In scenarios where

these entities are not part of the same application, STTP can be used to transfer the tokens securely between applications even if they are running on different hosts (e.g. the authorizing application is running on a mobile device).

2.2. Dynamic Password Provisioning

Passwords as shared secrets are very common for authentication and authorization. While in many cases they are pre-established, STTP can be used to dynamically provision passwords for specific scenarios. An example scenario is the generation of one-time-passwords that can be used to access services over an untrusted link or using an untrusted device. For example, a user can use a trusted mobile device that connects over a secure low-bandwidth link to the authentication and authorization server to retrieve a (one time) token that he can use to access a service on a public Internet terminal that uses a non-secured WiFi connection. In such a scenario the application on the public terminal could use STTP to request a set of tokens from the user's device which in turn uses STTP to retrieve the tokens (after a successful authentication) from the authentication and authorization server.

3. Protocol details

To transfer a set of tokens a STTP client contacts a STTP server over an appropriate protocol and they exchanges a set of requests and responses. A request is a command string followed by a number of arguments and a response is a numeric code together with a textual representation followed by a number of arguments. Within one connection multiple requests and responses can be exchanged until either the client or the server terminate the connection.


```
GETTOKENS
Type: OAuth
OAuth-Grant-Type: authorization_code
OAuth-Client-Id: s6BhdRkqt3
OAuth-Client-Secret: gX1fBat3bV
OAuth-Code: i1WsRn1uB
```

Example of a request message

```
200 OK
Type: OAuth
OAuth-Access-Token: SlAV32hkKG
OAuth-Refresh-Token: 8xLOxBtZp8
Expires-In: 3600
```

Example of a response message

3.1. STTP over HTTP

STTP can be used over HTTP(s). When using STTP over HTTP(s) a command MUST be included in the entity-body of a HTTP "POST" request with the "application/x-www-form-urlencoded" content-type.

```
POST /resource HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

GETTOKENS
Type: Plain
Identifier: john.doe@example.com
```

STTP over HTTP example request

4. Token Types

STTP supports a number of different token types.

4.1. OAuth

OAuth tokens are OAuth access grant tokens that can be further used by an application to gain access to a resource that is protected using the OAuth protocol.

4.2. Generic Tokens

Generic tokens are sets of each an identifier token (e.g. a username) and an authentication token (e.g. a password) that be further used by an application to gain access to a username/password protected resource.

5. Authentication

STTP delegates server authentication to the (optionally) underlying TLS protocol. This document does describe its own methods to perform client authentication. Currently, client authentication via public/private key cryptography and password-based authentication using the Salted Challenge Response Authentication Mechanism (SCRAM) is supported.

5.1. Server authentication

Server authentication is performed using certificates during the TLS handshake (see [RFC5246], Section 7.4.2.). If the server did not authentication itself during the TLS handshake, the STTP client MAY terminate the connection if it requires server authentication.

5.2. Client authentication

If a server requires client authentication for a particular command, it will send a "Authentication Required" response message (see Section 8.4.2) on receiving a command message that does not include sufficient arguments to authenticate the client. The server MUST include a Server-Nonce argument in the response message.

5.2.1. Authentication Method Negotiation

To negotiate an authentication method, the client includes an Auth-Proposals argument in a command message which contains a list of supported authentication methods. If the client expects that an authentication will be required, for example, for the first command message of an STTP session, it can effectively save one protocol-round-trip by including an Auth-Proposals argument in the initial command message instead of waiting for an "Authentication Required" response message. A command message that includes an Auth-Proposals argument MUST also include a Client-Nonce argument.

When the server receives a command message that includes an Auth-Proposals argument and the server does indeed require client authentication, it MUST choose one of the proposals and return a "Authentication Required" response message with an appropriate Auth-

Scheme argument and the corresponding arguments that are required for the particular authentication method including the Client-Nonce. If the server does require client authentication but does not support any of the offered authentication methods it MUST respond with a "No Common Authentication Scheme" message. If the server does not require client authentication but the command message includes an Auth-Proposals argument, the server MUST ignore the argument.

When selecting an authentication methods the server SHOULD assume that the list of authentication proposals is ordered by client preference. The server SHOULD try to honor the client preferences by selecting the first authentication methods on the list that the server supports. However, operational or administrative directives MAY implement a server-side preference list (for example, by ranking the different authentication methods by the server load they induce) which CAN be used to override the client preferences.

5.2.2. Authenticated Request

To successfully complete an authentication, the client must compose an authenticated command message. An authenticated command message is a command message that includes a valid Client-Proof argument. The Client-Proof is an authentication method specific string that is computed over the particular command message. Please refer to the details of the authentication methods for the details how the Client-Proof is calculated. The input string for the calculation of the Client-Proof MUST be identical to the command message that the Client-Proof is valid for exclusive of the Client-Proof argument itself. Besides the Client-Proof argument, an authenticated command message MUST also include a Server-Nonce argument that reflects the Server-Nonce that was included by the server in the initial "Authentication Required" response message, a Client-Nonce argument, and a Reflection-Key argument.

5.2.3. SCRAM-SHA-1

SCRAM authentication is based on the Salted Challenge Response Authentication Mechanism (SCRAM) as described in [RFC5802] with the AuthMessage being the authenticated command message itself (without the Client-Proof argument). SCRAM is only used to authenticate the client.

5.2.4. RSA-SHA1

Authentication based on Section 8.2. of [RFC3447] (RSASSA-PKCS1-v1_5) with the message to be signed (M) being the authenticated command

message (without the Client-Proof argument).

6. Commands

6.1. GETTOKENS

The GETTOKENS command is used by an agent to request a token pair for a particular service URI.

6.1.1. Arguments

A client MUST include the following arguments in every GETTOKENS command:

- Service-URI

A client CAN include the following arguments in a GETTOKENS command:

- Auth-Proposals
- Client-Nonce
- Reflection-Key
- ...

7. Arguments

A STTP command can have a number of arguments. Arguments consist of an identifier followed by a colon (":") and the argument value. An argument is terminated by a line break. Basically 4.2 in [RFC2661]

7.1. Service-URI

The particular service URI that the current command pertains to. Within the context of a GETTOKENS command this is the service URI that the client is requesting a set of tokens for.

The Service-URI argument MUST follow the syntax described in [RFC3986]. When determining if a Service-URI matches a valid

resource, a server MUST follow the rules for URI comparison for the particular service context (e.g. [RFC2616] for HTTP resources). If no service specific rules exists, a server MUST use a case-sensitive octet-by-octet comparison of the given URI with the following exceptions:

- Comparisons of scheme names MUST be case-insensitive;

- Comparisons of host names MUST be case-insensitive;
- A port that is empty or not given is equivalent to the default port for the particular scheme;
- An empty path component is equivalent to the root path "/".

Characters that are not classified as "reserved" by [RFC3986] are equivalent to their corresponding percent-encoding ("%" HEXDIG HEXDIG).

7.2. Identifier

The identifier token (e.g. a username) of the entity that the current exchange is intended for (either a user or another STTP agent).

7.3. Server-Nonce

A nonce value

7.4. Client-Nonce

A nonce value

7.5. Reflection-Key

To prevent man-in-the-middle attacks during authenticated requests, STTP introduces a Reflection-Key. The Reflection-Key argument is included in an authenticated request and reflects the server that the client is talking to. If a server receives a Reflection-Key that doesn't match it's own properties, it MUST terminate the connection. The actual value of the Reflection-Key is dependent on the protocol

that STTP is run over.

7.5.1. TLS-based transport

When STTP is run over a TLS-based protocol such as HTTPS the Reflection-Key argument is the SHA-1 fingerprint of the public key that was used to establish the TLS connection.

7.5.2. Other transport

When STTP is run over other transport protocols the Reflection-Key argument is the SHA-1 hash of the IP-Address and port number that was used to initiate the connection.

7.5.3. Manual verification

In n-party scenarios where a STTP server is used to request a token on behalf on another client (e.g. a mobile device is used to request a token for an untrusted terminal) the STTP server and the STTP client SHOULD present the user with a human readable form of the Reflection-Key. This allows the user to manually verify that both Reflection-Keys match and to prevent a man-in-the-middle attack.

7.6. Auth-Scheme

The authentication scheme that is used in the request.

7.7. Client-Proof

The client-side result of an authentication run.

7.8. Auth-Proposals

Comma separated list of authentication schemes that are supported by the client to perform client authentication. MUST be included in a request that follows a "Authentication Required" response for the same service URI.

7.9. Iteration-Count

Integer value

7.10. Message

A message which contains more verbose information and that is intended to be displayed to the end user.

7.11. Type

The type of tokens that are being transferred.

- PLAIN
- OAuth

7.11.1. Plain

If the Type argument is set to Plain, the tokens that are being requested/transferred are generic tokens.

7.11.2. OAuth

If the Type argument is set to Plain, the tokens that are being requested/transferred are OAuth tokens.

7.12. Expires-In

Number of seconds the transferred tokens are still valid for.

7.13. OAuth-Grant-Type

7.14. OAuth-Client-Id

7.15. OAuth-Client-Secret

7.16. OAuth-Code

7.17. OAuth-Access-Token

7.18. OAuth-Refresh-Token

7.19. IToken

Generic identifier token (e.g. username).

7.20. AToken

Generic authentication token (e.g. password).

8. Response Codes

A server's response starts with a response code that reflects the outcome of the last client request. The available response codes are described in this section including the arguments that are valid for each response.

8.1. Informational 1xx

Currently no use.

8.2. Successful 2xx

8.2.1. 200 OK

8.3. Redirection 3xx

8.4. Client Error 4xx

8.4.1. 400 Bad Request

8.4.2. 401 Authentication Required

8.4.3. 402 Not Responsible

8.4.4. 403 Authentication Failed

8.4.5. 404 No Common Authentication Scheme

8.5. Server Error 5xx

8.5.1. 501 Not Implemented

9. Security considerations

9.1. Token security

Currently STTP relies on the transport protocol to protect the tokens. STTP MUST be used over a secure transport protocol such as HTTPS if the tokens are to be protected from eavesdropping.

9.2. Man-in-the-middle attacks

The goal of an adversary in STTP is to obtain a valid set of tokens pair which he can use to authenticate as the user or to obtain the user's credentials themselves. There are two possible attacks if an adversary successfully tricks a client to connect to a fraudulent server: either the adversary continues with a phishing attack with the goal to obtain user credentials or he switches to a man-in-the-middle attack to gain access to a valid set of tokens that are being transferred. In case he continues the phishing attack the fraudulent server is setup to successfully complete the user authentication even if the credentials cannot be verified. However, STTP only supports a very selective number of authentication methods that do not rely on shared secrets. Neither of the methods supported provide the party that a user is authenticating to with any data that can be used to impersonate the user. In case the attacker switches to a man-in-the-middle attack the STTP authentication that is being relayed from the attacker to the proper STTP server will fail since the Reflection-Key attribute (c.p. Section 7.5) during the authentication doesn't match. However in cases where no user authentication is used, STTP relies on user interaction to manually compare the Reflection-Keys.

9.3. Privacy protection

To protect the users privacy against eavesdroppers, a secure transport protocol such as TLS-based transports must be used. If

such a secure transport is not used, STTP discloses privacy-relevant information such as the URI that tokens are being requested for and the username.

10. Normative References

[I-D.ietf-oauth-v2]
Hammer-Lahav, E., Recordon, D., and D. Hardt, "The OAuth 2.0 Protocol", draft-ietf-oauth-v2-10 (work in progress), July 2010.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2661] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, August 1999.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5802] Newman, C., Menon-Sen, A., Melnikov, A., and N. Williams, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms", RFC 5802, July 2010.

Authors' Addresses

Niklas Neumann (editor)
University of Goettingen
Computer Networks Group
Goldschmidtstrasse 7
Goettingen 37077
Germany

Email: neumann@cs.uni-goettingen.de

Florian Tegeler
University of Goettingen
Computer Networks Group
Goldschmidtstrasse 7
Goettingen 37077
Germany

Email: tegeler@cs.uni-goettingen.de

Xiaoming Fu
University of Goettingen
Computer Networks Group
Goldschmidtstrasse 7
Goettingen 37077
Germany

Email: fu@cs.uni-goettingen.de