TRAM
Internet-Draft
Intended status: Standards Track
Expires: October 17, 2015

T. Reddy
P. Patil
R. Ravindranath
Cisco
J. Uberti
Google

April 15, 2015

Session Traversal Utilities for NAT (STUN) Extension for Third Party Authorization
draft-ietf-tram-turn-third-party-authz-14

Abstract

This document proposes the use of OAuth 2.0 to obtain and validate ephemeral tokens that can be used for Session Traversal Utilities for NAT (STUN) authentication. The usage of ephemeral tokens ensures that access to a STUN server can be controlled even if the tokens are compromised.

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 October 17, 2015.

Table of Contents

1. Introduction

Session Traversal Utilities for NAT (STUN) [RFC5389] provides a mechanism to control access via "long-term" username/ password credentials that are provided as part of the STUN protocol. It is expected that these credentials will be kept secret; if the credentials are discovered, the STUN server could be used by unauthorized users or applications. However, in web applications like WebRTC [I-D.ietf-rtcweb-overview] where JavaScript uses the browser functionality to make real-time audio and/or video calls, Web conferencing, and direct data transfer, ensuring this secrecy is typically not possible.

To address this problem and the ones described in [RFC7376], this document proposes the use of third party authorization using OAuth 2.0 [RFC6749] for STUN. Using OAuth 2.0, a client obtains an

ephemeral token from an authorization server e.g. WebRTC server, and the token is presented to the STUN server instead of the traditional mechanism of presenting username/password credentials. The STUN server validates the authenticity of the token and provides required services. Third party authorization using OAuth 2.0 for STUN explained in this specification can also be used with Traversal Using Relays around NAT (TURN) [RFC5766].

2. Terminology

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 [RFC2119].

- WebRTC Server: A web server that supports WebRTC [I-D.ietf-rtcweb-overview].

- Access Token: OAuth 2.0 access token.

- mac_key: The session key generated by the authorization server. This session key has a lifetime that corresponds to the lifetime of the access token, is generated by the authorization server and bound to the access token.

- kid: An ephemeral and unique key identifier. The kid also allows the resource server to select the appropriate keying material for decryption.

Some sections in this specification show WebRTC server as the authorization server and client as the WebRTC client, however WebRTC is intended to be used for illustrative purpose only.

3. Solution Overview

STUN client knows that it can use OAuth 2.0 with the target STUN server either through configuration or when it receives the new STUN attribute THIRD-PARTY-AUTHORIZATION in the error response with an error code of 401(Unauthorized).

This specification uses the token type 'Assertion' (aka self-contained token) described in [RFC6819] where all the information necessary to authenticate the validity of the token is contained within the token itself. This approach has the benefit of avoiding a protocol between the STUN server and the authorization server for token validation, thus reducing latency. The content of the token is opaque to the client. The client embeds the token within a STUN request sent to the STUN server. Once the STUN server has determined the token is valid, its services are offered for a determined period

of time. Access token issued by the authorization server is explained in Section 6.2. OAuth 2.0 in [RFC6749] defines four grant types. This specification uses the OAuth 2.0 grant type "Implicit" explained in section 1.3.2 of [RFC6749] where the client is issued an access token directly. The string 'stun' is defined by this specification for use as the OAuth scope parameter (see section 3.3 of [RFC6749]) for the OAuth token.

The exact mechanism used by a client to obtain a token from the OAuth 2.0 authorization server is outside the scope of this document. Appendix B provides an example deployment scenario of interaction between the client and authorization server to obtain a token.

4. Obtaining a Token Using OAuth

A STUN client needs to know the authentication capability of the STUN server before deciding to use third party authorization. A STUN client initially makes a request without any authorization. If the STUN server supports third party authorization, it will return an error message indicating that the client can authorize to the STUN server using an OAuth 2.0 access token. The STUN server includes an ERROR-CODE attribute with a value of 401 (Unauthorized), a nonce value in a NONCE attribute and a SOFTWARE attribute that gives information about the STUN server's software. The STUN server also includes the additional STUN attribute THIRD-PARTY-AUTHORIZATION signaling the STUN client that the STUN server supports third party authorization.

Note: An implementation may choose to contact the authorization server to obtain a token even before it makes a STUN request, if it knows the server details before hand. For example, once a client has learnt that a STUN server supports third party authorization from a authorization server, the client can obtain the token before making subsequent STUN requests.

4.1. Key Establishment

In this model the STUN server would not authenticate the client itself but would rather verify whether the client knows the session key associated with a specific access token. Example of this approach can be found with the OAuth 2.0 Proof-of-Possession (PoP) Security Architecture [I-D.ietf-oauth-pop-architecture]. The authorization server shares a long-term secret (K) with the STUN server. When the client requests an access token the authorization server creates a fresh and unique session key (mac_key) and places it into the token encrypted with the long term secret. Symmetric cryptography MUST be chosen to ensure that the size of encrypted token is not large because usage of asymmetric cryptography will

result in large encrypted tokens which may not fit into a single STUN message.

The STUN server and authorization server can establish a symmetric key (K) and certain authenticated encryption algorithm, using an out of band mechanism. The STUN and authorization servers MUST establish K over an authenticated secure channel. If Authenticated Encryption with AES-CBC and HMAC-SHA (defined in [I-D.mcgrew-aead-aes-cbc-hmac-sha2]) is used then the AS-RS and AUTH keys will be derived from K. The AS-RS key is used for encrypting the self-contained token and the message integrity of the encrypted token is calculated using the AUTH key. If Authenticated Encryption with Associated Data (AEAD) algorithm defined in [RFC5116] is used then there is no need to generate the AUTH key and AS-RS key will have the same value as K.

The procedure for establishment of the symmetric key is outside the scope of this specification, and this specification does not mandate support of any given mechanism. Section 4.1.1 and Section 4.1.2 show examples of mechanisms that can be used.

4.1.1. HTTP interactions

The STUN and AS servers could choose to use REST API over HTTPS to establish a symmetric key. HTTPS MUST be used for mutual authentication and confidentiality. To retrieve a new symmetric key, the STUN server makes an HTTP GET request to the authorization server, specifying STUN as the service to allocate the symmetric keys for, and specifying the name of the STUN server. The response is returned with content-type "application/json", and consists of a JavaScript Object Notation (JSON) [RFC7159] object containing the symmetric key.

```
   Request
   -------

   service - specifies the desired service (turn)
   name    -  STUN server name be associated with the key

   example: GET /?service=stun&name=turn1@example.com


   Response
   --------

   k - Long-term key (K)
   exp - identifies the time after which the key expires.


   example:
   {
      "k" :
   "ESIzRFVmd4iZABEiM0RVZgKn6WjLaTC1FXAghRMVTzkBGNaaN496523WIISKerLi",
      "exp" : 1300819380,
      "kid" :"22BIjxU93h/IgwEb"
      "enc" : A256GCMKW
     }
```

The authorization server must also signal kid to the STUN server which will be used to select the appropriate keying material for decryption. The parameter "k" is defined in Section 6.4.1 of [I-D.ietf-jose-json-web-algorithms], "enc" is defined in Section 4.1.2 of [I-D.ietf-jose-json-web-encryption], "kid" is defined in Section 4.1.4 of [I-D.ietf-jose-json-web-signature] and "exp" is defined in Section 4.1.4 of [I-D.ietf-oauth-json-web-token]. A256GCMKW and other authenticated encryption algorithms are defined in [I-D.ietf-jose-json-web-algorithms]. A STUN server and authorization server implementation MUST support A256GCMKW as the authenticated encryption algorithm.

If A256CBC-HS512 defined in [I-D.ietf-jose-json-web-algorithms] is used then the AS-RS and AUTH keys are derived from K using the mechanism explained in section 5.2.2.1 of [I-D.ietf-jose-json-web-algorithms]. In this case AS-RS key length must be 256-bit, AUTH key length must be 256-bit (section 2.6 of [RFC4868]).

4.1.2. Manual provisioning

The STUN and AS servers could be manually configured with a symmetric key (K), authenticated encryption algorithm and kid.

Note : The mechanism specified in Section 4.1.2 requires configuration to change the symmetric key (K) and/or authenticated encryption algorithm. Hence a STUN server and authorization server implementation SHOULD support REST explained in Section 4.1.1.

5. Forming a Request

When a STUN server responds that third party authorization is required, a STUN client re-attempts the request, this time including access token and kid values in ACCESS-TOKEN and USERNAME STUN attributes. The STUN client includes a MESSAGE-INTEGRITY attribute as the last attribute in the message over the contents of the STUN message. The HMAC for the MESSAGE-INTEGRITY attribute is computed as described in section 15.4 of [RFC5389] where the mac_key is used as the input key for the HMAC computation. The STUN client and server will use the mac_key to compute the message integrity and do not perform MD5 hash on the credentials.

6. STUN Attributes

The following new STUN attributes are introduced by this specification to accomplish third party authorization.

6.1. THIRD-PARTY-AUTHORIZATION

This attribute is used by the STUN server to inform the client that it supports third party authorization. This attribute value contains the STUN server name. The STUN server may have tie-ups with multiple authorization servers and vice versa, so the client MUST provide the STUN server name to the authorization server so that it can select the appropriate keying material to generate the self-contained token. The THIRD-PARTY-AUTHORIZATION attribute is a comprehension-optional attribute (see Section 15 from [RFC5389]). If the client is able to comprehend THIRD-PARTY-AUTHORIZATION it MUST ensure that third party authorization takes precedence over first party authentication (explained in section 10 of [RFC5389]). If the client does not support or is not capable of doing third party authorization then it defaults to first party authentication.

6.2. ACCESS-TOKEN

The access token is issued by the authorization server. OAuth 2.0 does not impose any limitation on the length of the access token but if path MTU is unknown then STUN messages over IPv4 would need to be less than 548 bytes (Section 7.1 of [RFC5389]). The access token length needs to be restricted to fit within the maximum STUN message size. Note that the self-contained token is opaque to the client and the client MUST NOT examine the token. The ACCESS-TOKEN attribute is a comprehension-required attribute (see Section 15 from [RFC5389]).

The token is structured as follows:

```
struct {
    uint16_t nonce_length;
    opaque  nonce[nonce_length];
    opaque {
        uint16_t key_length;
        opaque mac_key[key_length];
        uint64_t timestamp;
        uint32_t lifetime;
    } encrypted_block;
} token;
```

Figure 1: Self-contained token format

Note: uintN_t means an unsigned integer of exactly N bits. Single-byte entities containing uninterpreted data are of type opaque. All values in the token are stored in network byte order.

The associated data (A) MUST be the STUN server name. This ensures that the client does not use the same token to gain illegal access to other STUN servers provided by the same administrative domain i.e., when multiple STUN servers in a single administrative domain share the same symmetric key with an authorization server.

The fields are described below:

nonce_length: Length of the nonce field. The length of nonce for authenticated encryption with additional data (AEAD) algorithms is explained in [RFC5116].

Nonce: Nonce (N) formation is explained in section 3.2 of [RFC5116].

key_length: Length of the session key in octets. Key length of 160-bits MUST be supported (i.e., only 160-bit key is used by HMAC-SHA-1 for message integrity of STUN message). The key length

facilitates the hash agility plan discussed in section 16.3 of [RFC5389].

mac_key: The session key generated by the authorization server.

timestamp: 64-bit unsigned integer field containing a timestamp. The value indicates the time since January 1, 1970, 00:00 UTC, by using a fixed point format. In this format, the integer number of seconds is contained in the first 48 bits of the field, and the remaining 16 bits indicate the number of 1/64K fractions of a second (Native format - Unix).

lifetime: The lifetime of the access token, in seconds. For example, the value 3600 indicates one hour. The lifetime value MUST be greater than or equal to the "expires_in" parameter defined in section 4.2.2 of [RFC6749], otherwise resource server could revoke the token but the client would assume that the token has not expired and would not refresh the token.

encrypted_block: The encrypted_block (P) is encrypted and authenticated using the symmetric long-term key established between the STUN server and the authorization server.

The AEAD encryption operation has four inputs: K , N, A, and P, as defined in section 2.1 of [RFC5116] and there is a single output a ciphertext C or an indication that the requested encryption operation could not be performed.

If AES_CBC_HMAC_SHA2 (explained in section 2.1 of [I-D.mcgrew-aead-aes-cbc-hmac-sha2])) is used then the encryption process is illustrated below. The ciphertext consists of the string S, with the string T appended to it. Here C and A denote Ciphertext and STUN server name respectively. The octet string AL (section 2.1 of [I-D.mcgrew-aead-aes-cbc-hmac-sha2]) is equal to the number of bits in A expressed as a 64-bit unsigned big endian integer.

- AUTH = initial authentication key length octets of K,
- AS-RS = final encryption key length octets of K,
- S = CBC-PKCS5-ENC(AS-RS, encrypted_block),
  - Initialization vector is set to zero because the encrypted_block in each access token will not be identical and hence will not result in generation of identical ciphertext.
- mac = MAC(AUTH, A || S || AL),

- T = initial T_LEN octets of mac,

- C = S || T.

The entire token i.e., the 'encrypted_block' is base64 encoded (see section 4 of [RFC4648]) and the resulting access token is signaled to the client.

7. STUN Server Behaviour

The STUN server, on receiving a request with ACCESS-TOKEN attribute, performs checks listed in section 10.2.2 of [RFC5389] in addition to the following steps to verify that the access token is valid:

- STUN server selects the keying material based on kid signalled in the USERNAME attribute.

- The AEAD decryption operation has four inputs: K, N, A, and C, as defined in section 2.2 of [RFC5116]. AEAD decryption algorithm has only a single output, either a plaintext or a special symbol FAIL that indicates that the inputs are not authentic. If authenticated decrypt operation returns FAIL then the STUN server rejects the request with an error response 401 (Unauthorized).

- If AES_CBC_HMAC_SHA2 is used then the final T_LEN octets are stripped from C. It performs the verification of the token message integrity by calculating HMAC over the the STUN server name, the encrypted portion in the self-contained token and the AL using AUTH key and if the resulting value does not match the mac field in the self-contained token then it rejects the request with an error response 401 (Unauthorized).

- STUN server obtains the mac_key by retrieving the content of the access token (which requires decryption of the self-contained token using the AS-RS key).

- The STUN server verifies that no replay took place by performing the following check:

  * The access token is accepted if the timestamp field (TS) in the self-contained token is recent enough to the reception time of the STUN request (RDnew) using the following formula: Lifetime + Delta > abs(RDnew - TS). The RECOMMENDED value for the allowed Delta is 5 seconds. If the timestamp is NOT within the boundaries then the STUN server discards the request with error response 401 (Unauthorized).

- The STUN server uses the mac_key to compute the message integrity over the request and if the resulting value does not match the contents of the MESSAGE-INTEGRITY attribute then it rejects the request with an error response 401 (Unauthorized).

- If all the checks pass, the STUN server continues to process the request. Any response generated by the server MUST include the MESSAGE-INTEGRITY attribute, computed using the mac_key.

If a STUN server receives an ACCESS-TOKEN attribute unexpectedly (because it had not previously sent out a THIRD-PARTY-AUTHORIZATION), it will respond with an error code of 420 (Unknown Attribute) as specified in Section 7.3.1 of [RFC5389].

8. STUN Client Behaviour

- The client looks for the MESSAGE-INTEGRITY attribute in the response. If MESSAGE-INTEGRITY is absent or the value computed for message integrity using mac_key does not match the contents of the MESSAGE-INTEGRITY attribute then the response MUST be discarded.

- If the access token expires then the client MUST obtain a new token from the authorization server and use it for new STUN requests.

9. Usage with TURN

Traversal Using Relay NAT (TURN) [RFC5766] an extension to the STUN protocol is often used to improve the connectivity of P2P applications. TURN ensures that a connection can be established even when one or both sides is incapable of a direct P2P connection. However, as a relay service, it imposes a nontrivial cost on the service provider. Therefore, access to a TURN service is almost always access-controlled. In order to achieve third party authorization, a resource owner e.g. WebRTC server, authorizes a TURN client to access resources on the TURN server.

Consider the following example that illustrates the use of OAuth 2.0 to achieve third party authorization for TURN. In this example, a resource owner i.e., WebRTC server, authorizes a TURN client to access resources on a TURN server.

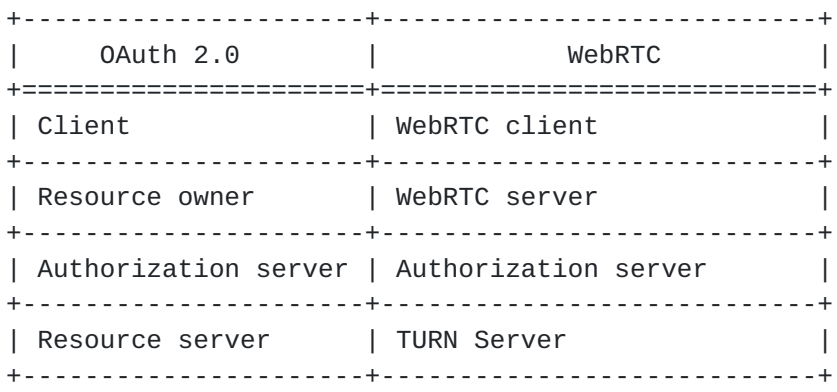
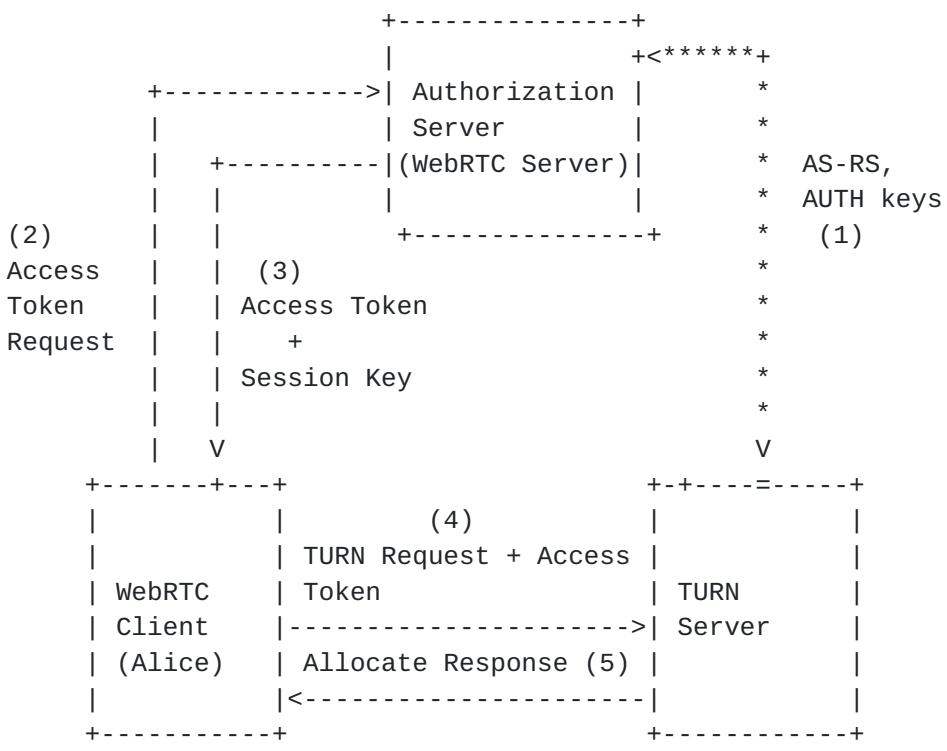


Figure 2: OAuth terminology mapped to WebRTC terminology

Using the OAuth 2.0 authorization framework, a WebRTC client (third-party application) obtains limited access to a TURN (resource server) on behalf of the WebRTC server (resource owner or authorization server). The WebRTC client requests access to resources controlled by the resource owner (WebRTC server) and hosted by the resource server (TURN server). The WebRTC client obtains access token, lifetime, session key and kid. The TURN client conveys the access token and other OAuth 2.0 parameters learnt from the authorization server to the TURN server. The TURN server obtains the session key from the access token. The TURN server validates the token, computes the message integrity of the request and takes appropriate action i.e, permits the TURN client to create allocations. This is shown in an abstract way in Figure 3.



User : Alice
****: Out-of-Band Long-Term Key Establishment

Figure 3: Interactions

In the below figure, the client sends an Allocate request to the server without credentials. Since the server requires that all requests be authenticated using OAuth 2.0, the server rejects the request with a 401 (Unauthorized) error code and STUN attribute THIRD-PARTY-AUTHORIZATION. The WebRTC client obtains access token from the WebRTC server and then tries again, this time including access token. This time, the server validates the token, accepts the Allocate request and returns an Allocate success response containing (amongst other things) the relayed transport address assigned to the allocation.

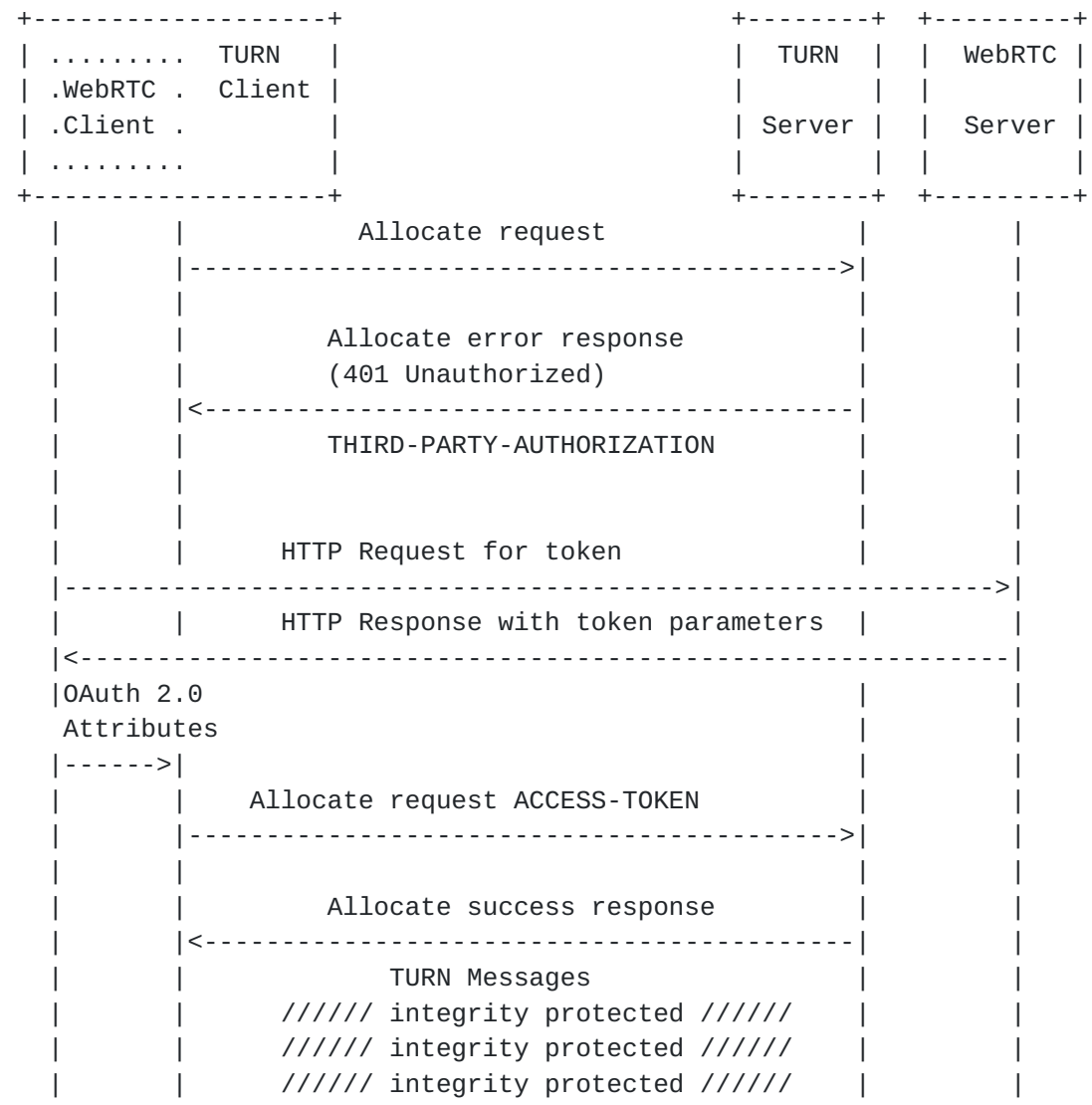


Figure 4: TURN Third Party Authorization

Changes specific to TURN are listed below:

- The access token can be reused for multiple Allocate requests to the same TURN server. The TURN client MUST include the ACCESS-TOKEN attribute only in Allocate and Refresh requests. Since the access token is valid for a specific period of time, the TURN server can cache it so that it can check if the access token in a new allocation request matches one of the cached tokens and avoids the need to decrypt the token.

- The lifetime provided by the TURN server in the Allocate and Refresh responses MUST be less than or equal to the lifetime of the token. It is RECOMMENDED that the TURN server calculate the maximum allowed lifetime value using the formula:

lifetime + Delta - abs(RDnew - TS)

The RECOMMENDED value for the allowed Delta is 5 seconds.

- If the access token expires then the client MUST obtain a new token from the authorization server and use it for new allocations. The client MUST use the new token to refresh existing allocations. This way client has to maintain only one token per TURN server.

10. Operational Considerations

The following operational considerations should be taken into account:

- Each authorization server should maintain the list of STUN servers for which it will grant tokens, and the long-term secret shared with each of those STUN servers.

- If manual configuration (Section 4.1.2) is used to establish symmetric keys, the necessary information which includes long-term secret (K) and authenticated encryption algorithm have to be configured on each authorization server and STUN server for each kid. The client obtains the session key and HMAC algorithm from the authorization server in company with the token.

- When a STUN client sends a request to get access to a particular STUN server (S) the authorization server must ensure that it selects the appropriate kid, access-token depending on the server S.

11. Security Considerations

When OAuth 2.0 is used the interaction between the client and the authorization server requires Transport Layer Security (TLS) with a ciphersuite offering confidentiality protection and the guidance given in [I-D.ietf-uta-tls-bcp] must be followed to avoid attacks on TLS. The session key MUST NOT be transmitted in clear since this would completely destroy the security benefits of the proposed scheme. An attacker trying to replay message with ACCESS-TOKEN attribute can be mitigated by frequent changes of nonce value as discussed in section 10.2 of [RFC5389]. The client may know some (but not all) of the token fields encrypted with a unknown secret key and the token can be subjected to known-plaintext attack, but AES is secure against this attack.

An attacker may remove the THIRD-PARTY-AUTHORIZATION STUN attribute from the error message forcing the client to pick first party

authentication, this attack may be mitigated by opting for Transport Layer Security (TLS) [RFC5246] or Datagram Transport Layer Security (DTLS) [RFC6347] as a transport protocol for Session Traversal Utilities for NAT (STUN), as defined in [RFC5389]and [RFC7350].

Threat mitigation discussed in section 5 of [I-D.ietf-oauth-pop-architecture] and security considerations in [RFC5389] are to be taken into account.

12. IANA Considerations

[Paragraphs below in braces should be removed by the RFC Editor upon publication]

[IANA is requested to add the following attributes to the STUN attribute registry [iana-stun], The THIRD-PARTY-AUTHORIZATION attribute requires that IANA allocate a value in the "STUN attributes Registry" from the comprehension-optional range (0x8000-0xBFFF)]

This document defines the THIRD-PARTY-AUTHORIZATION STUN attribute, described in Section 6. IANA has allocated the comprehension-optional codepoint TBD for this attribute.

[The ACCESS-TOKEN attribute requires that IANA allocate a value in the "STUN attributes Registry" from the comprehension-required range (0x0000-0x3FFF)]

This document defines the ACCESS-TOKEN STUN attribute, described in Section 6. IANA has allocated the comprehension-required codepoint TBD for this attribute.

13. Acknowledgements

Authors would like to thank Dan Wing, Pal Martinsen, Oleg Moskalenko, Charles Eckel, Spencer Dawkins, Hannes Tschofenig, Yaron Sheffer, Tom Taylor, Christer Holmberg, Pete Resnick, Kathleen Moriarty, Richard Barnes, Stephen Farrell and Alissa Cooper for comments and review. The authors would like to give special thanks to Brandon Williams for his help.

Thanks to Oleg Moskalenko for providing token samples in the Appendix section.

14. References

14.1. Normative References

[I-D.ietf-jose-json-web-algorithms]
Jones, M., "JSON Web Algorithms (JWA)", draft-ietf-jose-json-web-algorithms-40 (work in progress), January 2015.

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

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC4868] Kelly, S. and S. Frankel, "Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec", RFC 4868, May 2007.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, January 2008.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[iana-stun]
IANA, , "IANA: STUN Attributes", April 2011, <http://www.iana.org/assignments/stun-parameters/stun-parameters.xml>.

14.2. Informative References

[I-D.ietf-jose-json-web-encryption]
Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption-40 (work in progress), January 2015.

[I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-41 (work in progress), January 2015.

[I-D.ietf-oauth-json-web-token]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token-32 (work in progress), December 2014.

[I-D.ietf-oauth-pop-architecture]
Hunt, P., Richer, J., Mills, W., Mishra, P., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession (PoP) Security Architecture", draft-ietf-oauth-pop-architecture-01 (work in progress), March 2015.

[I-D.ietf-oauth-pop-key-distribution]
Bradley, J., Hunt, P., Jones, M., and H. Tschofenig, "OAuth 2.0 Proof-of-Possession: Authorization Server to Client Key Distribution", draft-ietf-oauth-pop-key-distribution-01 (work in progress), March 2015.

[I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-13 (work in progress), November 2014.

[I-D.ietf-tram-stunbis]
Petit-Huguenin, M., Salgueiro, G., Rosenberg, J., Wing, D., Mahy, R., and P. Matthews, "Session Traversal Utilities for NAT (STUN)", draft-ietf-tram-stunbis-04 (work in progress), March 2015.

[I-D.ietf-uta-tls-bcp]
Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of TLS and DTLS", draft-ietf-uta-tls-bcp-11 (work in progress), February 2015.

[I-D.mcgrew-aead-aes-cbc-hmac-sha2]
McGrew, D., Foley, J., and K. Paterson, "Authenticated Encryption with AES-CBC and HMAC-SHA", draft-mcgrew-aead-aes-cbc-hmac-sha2-05 (work in progress), July 2014.

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

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC6819] Lodderstedt, T., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, January 2013.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[RFC7350] Petit-Huguenin, M. and G. Salgueiro, "Datagram Transport Layer Security (DTLS) as Transport for Session Traversal Utilities for NAT (STUN)", RFC 7350, August 2014.

[RFC7376] Reddy, T., Ravindranath, R., Perumal, M., and A. Yegin, "Problems with Session Traversal Utilities for NAT (STUN) Long-Term Authentication for Traversal Using Relays around NAT (TURN)", RFC 7376, September 2014.

Appendix A. Sample tickets

Input data (same for all samples below):

```
//STUN SERVER NAME
server_name = "blackdow.carleon.gov";

//Shared key between AS and RS

long_term_key = \x48\x47\x6b\x6a\x33\x32\x4b\x4a\x47\x69\x75\x79
                \x30\x39\x38\x73\x64\x66\x61\x71\x62\x4e\x6a\x4f
                \x69\x61\x7a\x37\x31\x39\x32\x33

//MAC key of the session (included in the token)
mac_key = \x5a\x6b\x73\x6a\x70\x77\x65\x6f\x69\x78\x58\x6d\x76\x6e
          \x36\x37\x35\x33\x34\x6d;

//length of the MAC key
mac_key_length  =  20;

//The timestamp field in the token
token_timestamp = 92470300704768;

//The lifetime of the token
token_lifetime = 3600;

//nonce for AEAD when AEAD is used
aead_nonce = \x68\x34\x6a\x33\x6b\x32\x6c\x32\x6e\x34\x62\x35;
```

Sample:

1)

```
  token encryption algorithm = AEAD_AES_256_GCM
  token auth algorithm = N/A
```

```
Result:
AS_RS key (32 bytes) =
\x48\x47\x6b\x6a\x33\x32\x4b\x4a\x47\x69\x75\x79
\x30\x39\x38\x73\x64\x66\x61\x71\x62\x4e\x6a\x4f
\x69\x61\x7a\x37\x31\x39\x32\x33

AUTH key = N/A

Encrypted token (62 bytes = 34 + 16 + 12) =

\xd4\x86\x5c\x5d\x59\xfb\x3f\xe3\xf6\xf1\xd8\xc3\x22\xc2\x22\x26\x8d
\x2e\xf0\xbe\x2\x5b\xbd\x13\x49\x89\x6e\xa5\xc5\x51\xee\xee\x7f\xd9
\xe4\x41\xd7\xcb\x51\x20\x40\xcc\xc5\x53\x90\x2f\xdc\xbb\x8d\x53\x68
\x34\x6a\x33\x6b\x32\x6c\x32\x6e\x34\x62\x35
```

Figure 5: Sample tickets

Appendix B. Interaction between client and authorization server

Client makes an HTTP request to an authorization server to obtain a token that can be used to avail itself of STUN services. The STUN token is returned in JSON syntax [RFC7159], along with other OAuth 2.0 parameters like token type, key, token lifetime and kid defined in [I-D.ietf-oauth-pop-key-distribution].

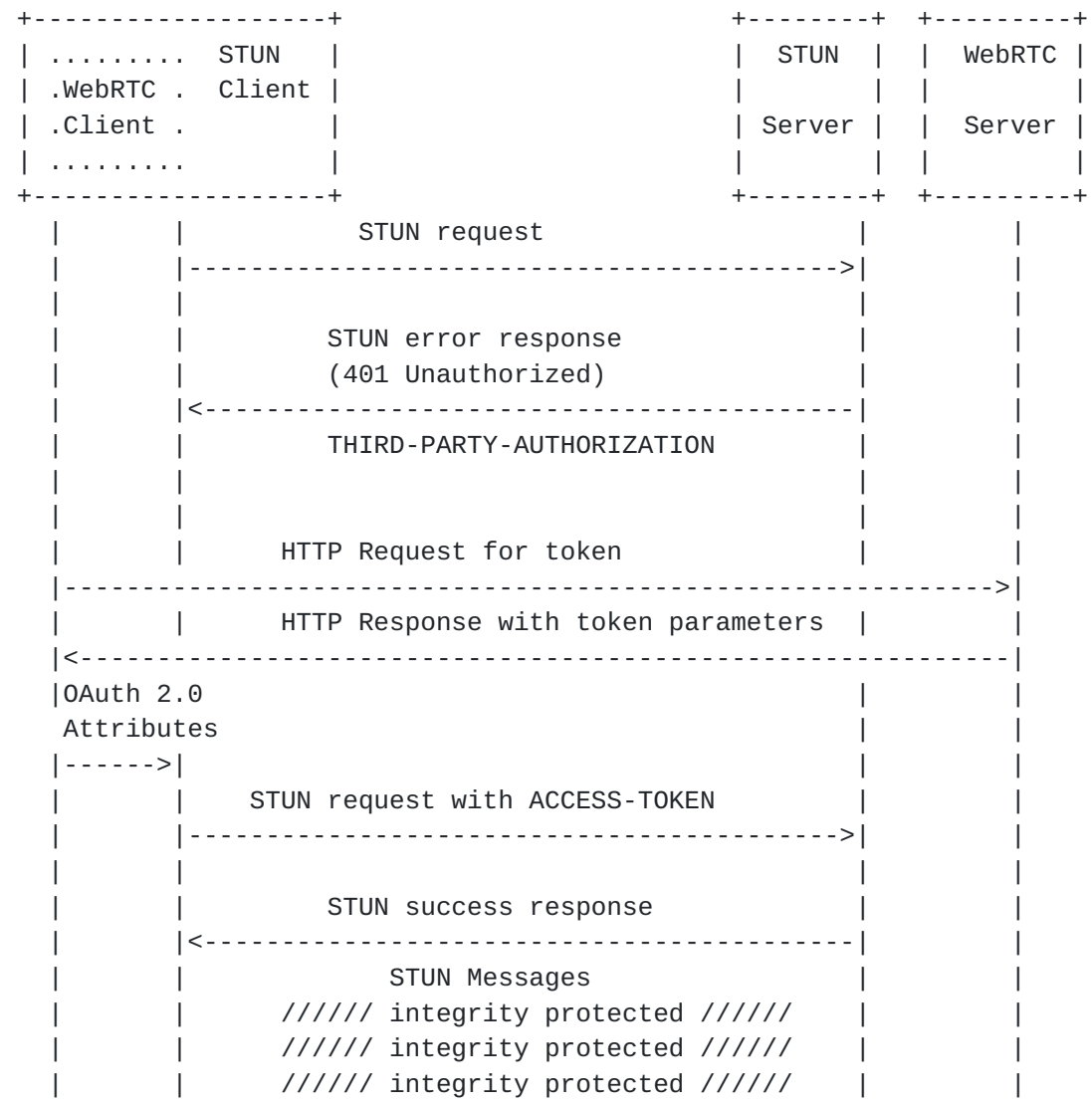


Figure 6: STUN Third Party Authorization

[I-D.ietf-oauth-pop-key-distribution] describes the interaction between the client and the authorization server. For example, the client learns the STUN server name "stun1@example.com" from THIRD-PARTY-AUTHORIZATION attribute value and makes the following HTTP request for the access token using transport-layer security (with extra line breaks for display purposes only):

```
HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
aud=stun1@example.com
timestamp=1361471629
grant_type=implicit
token_type=pop
alg=HMAC-SHA-1 HMAC-SHA-256-128
```

Figure 7: Request

[I-D.ietf-tram-stunbis] will support hash agility and accomplish this agility by conveying the HMAC algorithms supported by the STUN server along with a STUN error message to the client. The client then signals the intersection-set of algorithms supported by it and the STUN server to the authorization server in the 'alg' parameter defined in [I-D.ietf-oauth-pop-key-distribution]. The authorization server selects an HMAC algorithm from the list of algorithms the client provided and determines length of the mac_key based on the selected HMAC algorithm. Note that until STUN supports hash agility HMAC-SHA1 is the only valid hash algorithm that the client can signal to the authorization server and vice-versa.

If the client is authorized then the authorization server issues an access token. An example of successful response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
  "access_token":
"U2FsdGVkX18qJK/kkWmRcnfHglrVTJSpS6yU32kmHmOrfGyI3m1gQj1jRPsr0uBb
HctuycAgsfRX7nJW2BdukGyKMXSiNGNnBzigkAofP6+Z3vkJ1Q5pWbfSRroOkWBn",
  "token_type":"pop",
  "expires_in":1800,
  "kid":"22BIjxU93h/IgwEb",
  "key":"v51N62OM65kyMvfTI08O"
  "alg":HMAC-SHA-256-128
}
```

Figure 8: Response

Authors' Addresses

Tirumaleswar Reddy
Cisco Systems, Inc.
Cessna Business Park, Varthur Hobli
Sarjapur Marathalli Outer Ring Road
Bangalore, Karnataka 560103
India

Email: tireddy@cisco.com

Prashanth Patil
Cisco Systems, Inc.
Bangalore
India

Email: praspati@cisco.com

Ram Mohan Ravindranath
Cisco Systems, Inc.
Cessna Business Park,
Kadabeesanahalli Village, Varthur Hobli,
Sarjapur-Marathahalli Outer Ring Road
Bangalore, Karnataka 560103
India

Email: rmohanr@cisco.com

Justin Uberti
Google
747 6th Ave S
Kirkland, WA
98033
USA

Email: justin@uberti.name