

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: June 11, 2011

T. Hardjono, Ed.
MIT Kerberos Consortium
December 8, 2010

OAuth 2.0 support for the Kerberos V5 Authentication Protocol
draft-hardjono-oauth-kerberos-01

Abstract

This draft proposes an OAuth2.0 profile for Kerberos v5. We compare the Kerberos protocol flow with the OAuth protocol flow and as far as possible map the relevant parameters in Kerberos to OAuth parameters. We propose the use of the OAuth 2.0 message flows and its tokens to carry Kerberos TGTs and Service Tickets in an opaque manner.

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 June 11, 2011.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	Requirements Language	4
2.	Background: Kerberos protocol (v5)	4
2.1.	History of Kerberos	4
2.2.	Terminology	5
2.3.	Overview of the Kerberos (V5) protocol	6
	2.3.1. Advantages and disadvantages of the Kerberos paradigm	8
3.	Functional Comparison of OAuth 2.0 and Kerberos V5	9
3.1.	Client Authentication	9
	3.1.1. OAuth Client Authentication Request	9
	3.1.2. Kerberos Client Authentication Request	10
3.2.	Access Token Response	11
	3.2.1. OAuth Access Token Response	11
	3.2.2. Kerberos Service Ticket	12
3.3.	Resource Access Request	13
3.4.	Refresh Token	13
3.5.	Conclusions and Recommendations	15
4.	An OAuth 2.0 Profile for Kerberos (DRAFT)	16
4.1.	Design Assumptions	16
	4.1.1. Separation of Kerberos functions from OAuth functions	17
	4.1.2. Opaqueness of Kerberos Messages	17
	4.1.3. Resource Server as a Service Principal	17
4.2.	OAuth-Kerberos Architecture Overview	17
5.	Kerberized Token Exchange Service	18
6.	Acknowledgements	18
7.	IANA Considerations	19
8.	Security Considerations	19
9.	References	19
	9.1. Normative References	19
	9.2. Informative References	19
	Appendix A. Document History	20
	Author's Address	20

1. Introduction

There are a number of reasons why OAuth 2.0 support for the Kerberos protocol would be useful to the security industry. First, there is a large degree of similarity at the protocol design level and intended functions between OAuth 2.0 and Kerberos. There is also strong similarities in the entities involved in OAuth2.0 and Kerberos, albeit OAuth focuses on the application layer (above the HTTP layer). Access to a service requires the client to obtain an Access Token (or Service Ticket in Kerberos), while the Refresh Token (or Ticket Granting Ticket in Kerberos) is used to limit the number of re-authentications by the user seeking to use the same service.

Secondly, easily upwards of 60% of medium to large Enterprises today use the Kerberos protocol in one form or another (proprietary implementations or open source), for various authentication requirements. These range from user-logons at the operating system level, to Web-SSO and to server-to-server authentication at the backend infrastructure. Support for Kerberos is built into various operating systems and open-source distributions (e.g. Windows, RedHat, Open Solaris, Ubuntu, etc), and the majority of browsers support some Kerberos functions. Allowing organizations to place OAuth 2.0 servers as a front-end layer allows them to further re-use their existing Kerberos infrastructure, thereby allowing easier adoption of OAuth 2.0 within these organizations.

Thirdly, there are a number similarities in intended functions between Kerberos and OAuth 2.0. Any OAuth 2.0 function that can re-use Kerberos functionality and code may provide the developer with faster development time.

Finally, interoperability between OAuth2.0 and Kerberos is highly desired from a deployment perspective. There are a number of ways to achieve such interoperability. For example, OAuth2.0 opaque tokens and other structures can be used to transport Kerberos messages. Transporting or tunneling Kerberos above other protocols is not a new

idea. Various other protocols have previously been used to transport Kerberos message structures opaquely in order to satisfy certain security and deployment constraints. Examples include Kerberos within HTTP-Negotiate [[RFC4559](#)] and Kerberos over Web Services Security (over SOAP) [[WSS-Kerberos](#)].

The aim of this draft is to identify and define mechanisms that would allow a high degree of interoperability between OAuth 2.0 and Kerberos. As such, it is not the intent of the current work to replace OAuth 2.0, but instead to allow organizations to deploy OAuth 2.0 compliant Authorization Servers and Resource Servers, while retaining (and value-adding to) their Kerberos infrastructure in the

back-end.

In the next section we will provide an overview of the Kerberos (v5) protocol. The text of the discussion is intentionally kept at the same level of abstraction as the OAuth 2.0 Framework specification. This is to avoid the reader from having to read through [RFC4120](#) and to avoid the discussion from diving into the specific details and constraints of [RFC4120](#). The interested reader is directed to [RFC4120](#) for the details of the Kerberos authentication protocol.

[1.1](#). Requirements Language

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](#)].

[2](#). Background: Kerberos protocol (v5)

[2.1](#). History of Kerberos

The Kerberos authentication protocol [REF] was one of the earliest authentication protocols based purely on symmetric key cryptography. Founded on the Needham-Schroeder protocol [REF], Kerberos was designed as the authentication mechanism within Project Athena [REF] at MIT in the mid-1980s. In order to realize and implement the Needham-Schroeder protocol, a number of constructs were added to the basic Needham-Schroeder protocol to create Kerberos. Notable among these additions were the Kerberos ticket, and later the Ticket

Granting Service (TGS) as a separate function from the Authentication Server (AS) within the Key Distribution Center (KDC). The Kerberos V5 protocol was defined in 1993 within [RFC1510](#) (obsoleted), with the current authoritative definition being [RFC4210](#).

Kerberos is one of the few symmetric key based authentication protocols that has been well studied and scrutinized over the last two decades. The security of Kerberos is founded on the sharing of long-term symmetric keys (pair-wise between entities) and the usage of shorter term session keys between the Client and the Service. In summary, the Client and the KDC share a pair-wise unique symmetric key (long-term) that is used by both parties to communicate with data confidentiality and perform authentication based on proof-of-possession (POP) the correct keys. When a Client seeks to access services or resources at the Server, the Client must request a Service Ticket to the KDC specifically designated for the target Server. In order to prove to the Server that the Client is authorized by the KDC to access the Server, the KDC generates a Session Key and delivers it in an encrypted fashion to the Client and

Server (using their respective long-term keys which they share pair-wise with the KDC). When seeking access to the Server, the Client must prove possession of the session-key (to the Server) by encrypting a so called Authenticator structure using that session-key and sending it to the Server with the request. Successful decryption of the Authenticator structure by the Server indicates to the Server that the requestor is a legitimate Client who has been authenticated previously by the KDC. Here the Client must actually exercise its copy of the session-key to prove POP. In this manner, Kerberos is different from bearer-token protocols in which the client needs only present a token that was previously created by another entity.

[2.2.](#) Terminology

In this section we briefly summarize some of the terminology commonly used in the Kerberos literature, while providing the approximate OAuth 2.0 equivalent:

Client A process that makes use of a service or resource on behalf of a user. Note that in some cases a Server may itself be a client of some other server (e.g., a print server may be a client of a file server).

Authentication Server (AS) Authentication Server (AS) The service operating within the Key Distribution Center (KDC) that authenticates the client based on symmetric key cryptography. The client and the AS shares a long-term secret (long term symmetric key). The AS issues the client with a Ticket Granting Ticket (TGT) intended for a specific TGS (i.e. decipherable only by the intended TGS). The matching function in OAuth 2.0 is that of the token endpoint in the Authorization Server that issues the initial Access Token.

Authenticator A Kerberos record or structure containing information that can be shown to have been recently generated using the session key known only (shared) by the client and server. In Kerberos the authenticator is one way the client proves to the server that the client is in possession of the correct session key.

Ticket Granting Server (TGS) The service operating within the Key Distribution Center (KDC) that authenticates a client based on (a) the client's presentation of a legitimate Ticket Granting Ticket (TGT) issued by the AS and (b) the client's proof-of-possession of the correct symmetric key (used by the client to create the Authenticator portion of the TGS Request). The matching function in OAuth 2.0 is that of the token endpoint in the Authorization Server that issues the

Refresh Token.

Key Distribution Center (KDC) A Kerberos service that is comprised of the Authentication Server (AS) and Ticket Granting Server (TGS). The combined Kerberos AS and TGS could be matched to the OAuth 2.0 Authorization Server, where the TGS matches more closely the Refresh Token issuing function of the Authorization Server.

Principal A uniquely named client or server instance that participates in a network communication.

Server (or Kerberos Server) A particular Principal which provides a service or resource to clients. In OAuth 2.0 this is the Resource Server.

Session Key A temporary encryption key used between two principals, with a lifetime limited to the duration of a single Kerberos "session".

Sub-session Key A temporary encryption key used between two principals, selected and exchanged by the principals using the session key, and with a lifetime limited to the duration of a single association.

Ticket A record or data structure that helps a client authenticate itself to a server. It contains the client's identity, a session key, a timestamp and other information, all of which are sealed using the server's secret key. The term "service ticket" is often used to distinguish from the Ticket-Granting-Ticket (TGT) which is consumable only by the Ticket Granting Server (TGS). A ticket only serves to authenticate a client to a service when presented along with a fresh Authenticator. In OAuth 2.0 the closest structure to a ticket is the Access Token and Refresh Token, together with parameters accompany the token when transmitted.

[2.3.](#) Overview of the Kerberos (V5) protocol

The basic purpose of Kerberos is to provide access control of a client (user or host) to a service or resource. To simplify the current description, we will use the term "client" to mean the entity associated with the symmetric key shared with the Authentication Server (though the intent is clear that both human users and host computers can in fact be Kerberos principals).

The following summarizes the main message flows within Kerberos:

(A) Authentication Request (AS_REQ)

The Authentication Request (AS_REQ) message is sent by the Kerberos Client to the Authentication Server (AS) within the KDC.

(B) Authentication Response (AS_REP)

The AS/KDC returns a response (AS_REP) message to the Client that contains the encrypted Service Ticket and the encrypted

AS Response Part. These two parts contain the session-key that the Client and Service (Resource Server) will later use to secure their session.

(C) Service Request (AP_REQ)

The Client constructs the Authenticator structure that it encrypts using the session-key (obtained in the previous step). This Authenticator provides proof-of-possession (of the session-key) to the Service (Resource Server). The Client delivers both the Service Ticket (unmodified from the previous step) and the Authenticator within the AP_REQ message.

(D) Service Response (AP_REP)

The Service (Resource Server) obtains the AP_REQ message, decrypts the Session Ticket (created by the KDC) and extracts the session-key. It then uses the session-key to verify the Authenticator structure.

(E) Optional Ticket-Granting Service Request (TGS_REQ)

In order to prevent the Client from having to authenticate repeatedly to the KDC (e.g. due to short-life tickets), the Client can request a Ticket-Granting-Ticket (TGT) from the Ticket Granting Service (TGS), which is a special service operating at the KDC.

(F) Optional Ticket-Granting Service Response (TGS_REP)

If the TGS_REQ message is used, the TGS (instead of the AS) will return the Service Ticket destined for a given Service (Resource Server). Note that the Service Ticket life is shorter than the TGT life.

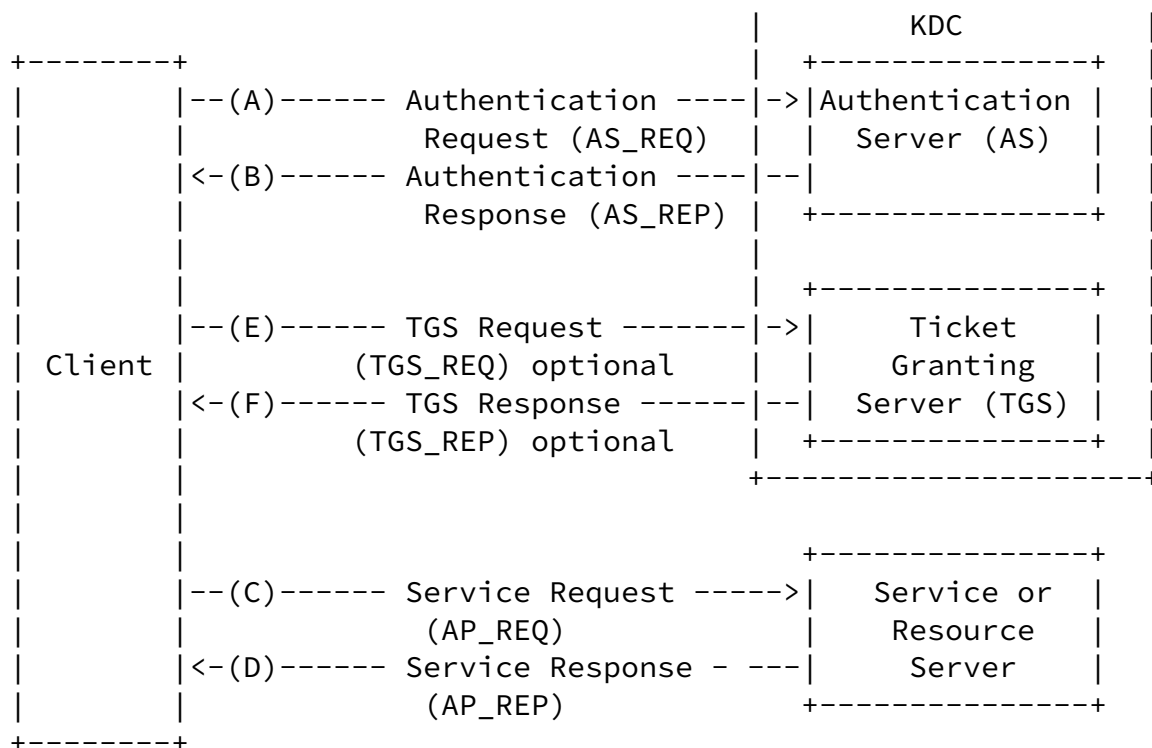


Figure 1

2.3.1. Advantages and disadvantages of the Kerberos paradigm

As an authentication protocol based on symmetric key cryptography, there a number of advantages as well as inherent limitations of Kerberos:

- o Limited impact of key compromise: The compromise of a client or server (or the keys in these entities) impacts only these entities. Other clients and servers are not impacted since they share a unique key with the KDC.
- o Speed of cryptographic operations: One of the original motivations behind Kerberos was the desire to use symmetric key ciphers due to their significant performance over public key ciphers.
- o Cross realm/domain authentication: Kerberos v5 addressed the requirement of cross-realm authentication and authorization by the introduction of cross-realm tickets. This provides the advantage of scaling Kerberos across realms (within one administrative boundary) and across realms in differing administrative boundaries.

- o Delegation: Kerberos v5 addressed supports the notion of delegation through the use of special tickets (referred to as proxiable tickets and forwardable tickets). In essence, delegation in Kerberos allows a Client to name another entity (e.g. a Server) within a ticket-request (to the KDC) with the explicit intent of allowing that entity to later request service-tickets in the name of the Client. Delegation can also be used to realize the notion of "transitive-trust" that may cross domain boundaries.
- o Compromise of KDC impacts other entities: Since the KDC shares a symmetric key pair-wise with every entity in the system (clients and server), the compromise of the KDC and the keys stored at the KDC renders the entire system compromised. Thus, additional security is required for keys stored at the KDC. Additional precautions, such as separating the Authentication Server (AS) into a more secure computing environment may reduce the risks associate with attacks.
- o Initial key establishment problem: Similar to many symmetric key based authentication protocols, Kerberos requires initial key establishment between a Client and the KDC, and between a Server and the KDC. Several methods to establish the initial shared key between entities have been proposed. These are referred to as "pre-authentication" protocols in the Kerberos literature, and includes a password-based method (defined as part of [RFC4210](#)), a public-key based approach (PKINIT [[RFC4556](#)]) using the Diffie-Hellman or RSA cryptosystems) as well as One-Time-passwords or tokens ([draft-ietf-krb-wg-otp-preauth-17](#)).

[3.](#) Functional Comparison of OAuth 2.0 and Kerberos V5

In this section we compare the intent/purpose of the OAuth 2.0 messages against those in Kerberos V5. The aim here is to understand the semantic similarities between OAuth 2.0 and Kerberos, and identifying the gaps needing solutions in order for OAuth 2.0 to support Kerberos V5. Another aspect of this exercise is also to understand the design decisions of Kerberos V5 over the last two decades, hopefully to inform a better design to OAuth 2.0 and other new protocols in the future.

[3.1.](#) Client Authentication

[3.1.1.](#) OAuth Client Authentication Request

The OAuth 2.0 protocol caters for a number of use-cases, though in many of these scenarios there are some common fundamental parameters

that are being exchanged with the Authorization Server.

In the Client Authentication exchange the token endpoint at the Authorization Server requires the client to provide the client identifier (`client_id`) and the optional client secret (`client_secret`). These two parameters can be delivered using a simple HTTP POST message or using an HTTP authentication scheme. How the client obtains these credentials are outside the scope of the OAuth 2.0 specification. Additional parameters that are communicated between the client and Authorization server:

- o Type of authentication (`user_agent`, `web_server`, `device_code`, `username`, `client_credentials`, `assertion` or `refresh`).
- o Client identity (`client_id`)
- o Client secret (`client_secret`)
- o Server identity (Host name)
- o Transport mechanism (e.g. HTTP POST)
- o Encoding method or content type (e.g. `urlencoded`)

The OAuth 2.0 specification, however, considers as out-of- scope the method used by the Authorization Server to authenticate the end-user. As such, the current OAuth 2.0 specification could be considered more as a framework for authentication and authorization, which needs to be instantiated with a specific authentication mechanism/method and which would therefore necessitate bindings to specific transports.

[3.1.2.](#) Kerberos Client Authentication Request

In Kerberos, the client initiates an authentication request by sending the `AS_REQ` message to the Authentication Server (AS). Ignoring for the moment the format of the `AS_REQ` message, the client sends the following parameters to the AS:

- o Message type

- o Kerberos protocol version number (v5)
- o Pre-authentication data (optional)
- o Client principal name
- o Client's realm

- o KDC principal name)
- o Start-time of requested ticket (optional)
- o Expiration time of requested ticket
- o Nonce generated by the client
- o Encryption algorithm desired
- o Client host address (optional)
- o Additional tickets (optional)

In Kerberos the client has the option of selecting the lifetime of tickets, though the KDC can set the upper bounds of the lifetimes according to some predefined policy. The client has to explicitly choose a unique random nonce for the purposes of detecting replays of the responses coming from the KDC.

Although [RFC4120](#) allows the Kerberos authentication request message to be sent in clear text, in practice it is beneficial to protect this message from unauthorized modifications in-transit. Similar to OAuth, the request could be sent over a secure channel such as TLS/SSL or within some tunneling mechanism (e.g. FAST tunnel).

[3.2.](#) Access Token Response

[3.2.1.](#) OAuth Access Token Response

In OAuth 2.0 the access token response from the Authorization Server carries a number of parameters explicitly and implicitly ([Section](#)

[3.3.2.1](#)):

- o Access Token
- o Duration of token (expires in seconds)
- o Optional Refresh Token
- o Access Token Secret (optional, determined by client)
- o Scope
- o Transport mechanism (e.g. HTTP 1.1)
- o Encoding method or content type (e.g. app/json)

The OAuth 2.0 specification currently does not specify the sizes of tokens and other values received from the Authorization Server.

[3.2.2](#). Kerberos Service Ticket

The Kerberos v5 equivalent to the OAuth 2.0 Access Token is that of the Service Ticket. The client can obtain a Service Ticket directly from the KDC using the Authentication Request (AS_REQ) message. The response coming back from the KDC contains a number of relevant parameters:

- o Message type
- o Kerberos protocol version number (v5)
- o Pre-authentication data (optional)
- o Client principal name
- o Client's realm
- o Service Ticket (encrypted structure)
- o Response Part (encrypted structure)

The two important structures inside the Kerberos response message are the Response Part structure that is intended for the Client, and the Service Ticket (ST) structure which the Client must forward unmodified to the Server (or Resource Server). Generally each of these structures carry a number of parameters in-turn, including (a) the identities of the Client and Server, (b) the lifetime of the Ticket and (c) the Session-Key to be used by the Client in requesting service or access to the Server (Resource Server).

Here, an important security feature is that the Response Part structure (containing the Session-Key) is encrypted by the KDC using the long-term key it shares with the Client. This ensures that only the authenticated Client can decrypt the Response Part and make use of the contained Session-Key. Similarly, the important parameters within the Service Ticket structure is encrypted by the KDC using the long term key it shares with the Server (i.e. Resource Server). This ensures that only the Client and Server know the given Session-Key, and thus allows the Server to later recognize an authentic service request from the Client (encrypted by the Client using that very same Session-Key).

In comparing the OAuth Access Token Response message with the Kerberos authentication response message containing a Service Ticket,

there are a number of interesting aspects:

- a. OAuth Access Token Secret vs Kerberos Session-Key: The OAuth 2.0 specification allows Clients to make authenticated protected resource requests using an access token with a matching secret by calculating a set of values and including them in the request using the "Authorization" header field. The OAuth 2.0 specification defines the "hmac-sha256" algorithm to verify this secret.

In contrast, in Kerberos the Session-Key is used by the Client to encrypt (not hash) the Authenticator structure that the Client creates. The encrypted Authenticator structure is then sent together with the Service Ticket (obtained from the KDC in the prior step) to the Server (or Resource Server).

Kerberos uses encryption instead of hashing only due to the sensitivity of the parameters exchanged between the Client and

KDC, and between the Client and Server.

- b. **Selectable Ciphers and Hash Functions:** Kerberos supports various ciphers and hash functions as negotiated upon between the Client and KDC at the initial authentication request.

[3.3.](#) Resource Access Request

Once the Client obtains an OAuth 2.0 Access Token, it delivers the Access Token to the Resource Server within an HTTP Request message. OAuth supports a number of use-cases which it supports (User-Agent Flow, Web Server Flow, Device Flow, Username/Password Flow, Client Credential Flow and Assertion Flow). These use-cases are in fact distinct from the fundamental authentication feature of OAuth 2.0 based on Access Tokens. The use-cases address the various deployment scenarios and technologies that are available today to implement them.

In Kerberos, in order to access the Server the Client must present both the Authenticator (created by the Client) and the Service Ticket (created by the KDC). These two structures when combined allow the Server (or Resource Server) to establish that the Client has been authenticated by the KDC. The Server does not have to interact with the KDC when validating the Service Ticket.

[3.4.](#) Refresh Token

In OAuth 2.0 the token refresh is used when the lifetime of an access token is shorter than the lifetime of the authorization grant ([Section 4](#)). It allows clients to obtain a new access token without

having to go through the OAuth authorization flow again (to the Authorization Server) or involve the resource owner.

The relevant parameters of the OAuth 2.0 Refresh Token request message are as follows:

- o Type of authentication (set to "refresh")
- o Client identity (client_id)
- o Client secret (if client issued a secret)

- o Refresh Token (refresh_token)
- o Secret Type (optional, defaults to bearer token if there is no secret)
- o Response format (one of "json", "xml", or "form").

In Kerberos, in order to avoid the Client from having to authenticate to the KDC each time it wishes to access a Server, the Client must first obtain a Ticket Granting Ticket (TGT) from the Ticket Granting Service (TGS) operating in the KDC. The Client then uses the TGT each time it requests a Service Ticket to access the Server. This can be done as a system process, without the need to prompt the user. In this sense, the Refresh Token is identical in purpose to the Kerberos TGT.

The message structure of the TGT request message is identical to the message structure of a Service Ticket request. Indeed, in Kerberos the Ticket Granting Service (TGS) is viewed in the same manner as the services offered by any Kerberos Server.

In Kerberos the AS Response (AS_REP) message carries a number of parameters, including those intended for consumption by the Ticket Granting Service (TGS). These are summarized as follows:

- o Message type
- o Kerberos protocol version number (v5)
- o Pre-authentication data (optional)
- o Client principal name
- o Client's realm

- o Ticket Granting Ticket (encrypted structure)
- o AS Response Part (encrypted structure)

The two important structures inside the AS_REP message are the AS Response Part structure that is intended for the client, and the Ticket Granting Ticket (TGT) structure which the client must forward unmodified to the Ticket Granting Service. Each of these structures carry a number of parameters in-turn, including (a) the identities of the client, TGS and Server, (b) the lifetime of the TGT and (c) the Session-key to be used by the Client in requesting the Service Ticket to the TGS.

Here, an important security feature is that the AS Response Part structure (containing a session-key) is encrypted by the AS using the long-term key it shares with the client. This ensures that only the authenticated client can decrypt the AS Response Part and make use of the contained session- key. Similarly, the important parameters within the TGT structure is encrypted by the AS using the long term key it shares with the TGS entity. This ensures that only the client and TGS know the given session-key, and thus allows the TGS to later recognize an authentic Service Ticket request from the client (encrypted by the client using that very same session-key).

It is important to note that obtaining a TGT is optional in Kerberos, and the client can in fact request a Service Ticket directly from the AS by simply designating the identity of the Server in the AS Request message. This may prove cumbersome in practice, since the client may need to newly authenticate itself to the AS each time it seeks to access the Server.

[3.5.](#) Conclusions and Recommendations

In this section we offer some observations and recommendations regarding OAuth 2.0, in the light of the various lessons learned from the development of the Kerberos authentication protocol over the last couple of decades:

[1] Refresh Tokens and Access Tokens

In Kerberos, the introduction of the Ticket Granting Service (TGS) as a special service on the KDC was done to address the issue of the end-user having to re-authenticate each time the user wanted to access the server (resource server). In practice, Kerberos is almost always deployed using the TGS where the TGT has a longer lifetime compared to the service-tickets requested using a given TGT.

In OAuth the cases for using a Refresh Token (as a means to extend the life of the Access Token) needs to be made clear

and explicit. Furthermore, Refresh Tokens need to be provably related to its parent Access Token, in such a way that a third-party entity (e.g. auditor) can verify the sequence of events/messages that has been logged by the Resource Server.

[2] Support for multiple authentication methods

Being a framework, OAuth2.0 currently does not (and should not) call-out specific authentication protocols. In order to gain broad adoption in various organizations, and to gain usage in higher assurance environments, OAuth needs to support a broad range of authentication protocols and methods, including username/passwords, OTP-tokens, Kerberos tickets, and others.

[3] Delegation support

A key feature of Kerberos that is used extensively in some deployments is delegation (or "constrained delegation" in the Kerberos literature). In Kerberos delegation is expressed using proxiable tickets and forwardable tickets, where a Kerberos client can request a ticket within which the client can explicitly name a server who will act as a delegate. This server will in-turn request service ticket on behalf of the client. Although delegation is not explicitly supported as a feature in the current OAuth 2.0 specification one could conceivably introduce the notion of a "delegation access token" that implements the same function as that in Kerberos.

[4.](#) An OAuth 2.0 Profile for Kerberos (DRAFT)

In this section we propose a "profile" of OAuth 2.0 for Kerberos that provides one approach for interoperability between OAuth2.0 and Kerberos. In this approach, the Kerberos messages are delivered using OAuth as a mechanism of transport between the relevant Kerberos entities.

A key aspect of this proposal is the representation of the Kerberos messages in the form of GSS-Tokens as defined in [RFC2743](#) and [RFC4121](#). This provides opaqueness of the Kerberos messages to the higher layer applications, which in this case are the OAuth entities operating at the HTTP layer.

[4.1.](#) Design Assumptions

[4.1.1.](#) Separation of Kerberos functions from OAuth functions

Since one of the main goals of OAuth 2.0 is to provide simplicity to the end-user, one of the design assumptions here is that the creation and consumption of Kerberos is performed by Kerberos entities (or components) that are distinct from OAuth entities. Thus, for example, although the OAuth Authorization Server could also be implemented as an enhanced Kerberos KDC, we assume that the KDC is distinct from the Authorization Server.

This architectural separation allows organizations already deploying a Kerberos infrastructure to layer OAuth as front- end service for authenticating users and OAuth clients, following the use-cases defined in the OAuth specification.

[4.1.2.](#) Opaqueness of Kerberos Messages

Following the approach used by HTTP-Negotiate ([RFC4559](#)) which introduced a new auth-scheme based on the SPNEGO GSSAPI-Tokens, in this document we also propose the use the GSSAPI-Tokens ([RFC2743](#)) for all Kerberos messages that will be over (through) through OAuth messages. The GSSAPI-Tokens is assumed to be encoded in some HTTP-friendly manner (e.g. Base64 encoded).

The approach of using Kerberos GSSAPI-Tokens ([RFC4121](#)) is also used in the Web Services Security standard, notably in the WSS Kerberos Token Profile 1.1 [[WSS-Kerberos](#)]. There the Kerberos AP_REQ message (wrapped GSS-Token) is delivered by a client to a service over WSS (over SOAP).

[4.1.3.](#) Resource Server as a Service Principal

In order to achieve some degree of interoperability, we assume that the OAuth Resource Server is also a service principal in the traditional Kerberos sense. This means (among others) that (a) the Resource Server shares a long-term key with the KDC, and (b) that it is able to consume and process Service Tickets in the usual Kerberos manner. How a Resource Server implements the Kerberos Service is outside the scope of the OAuth 2.0 specification and of this current document.

4.2. OAuth-Kerberos Architecture Overview

The overall architecture underlying the OAuth 2.0 profile for Kerberos is shown in Figure 2.

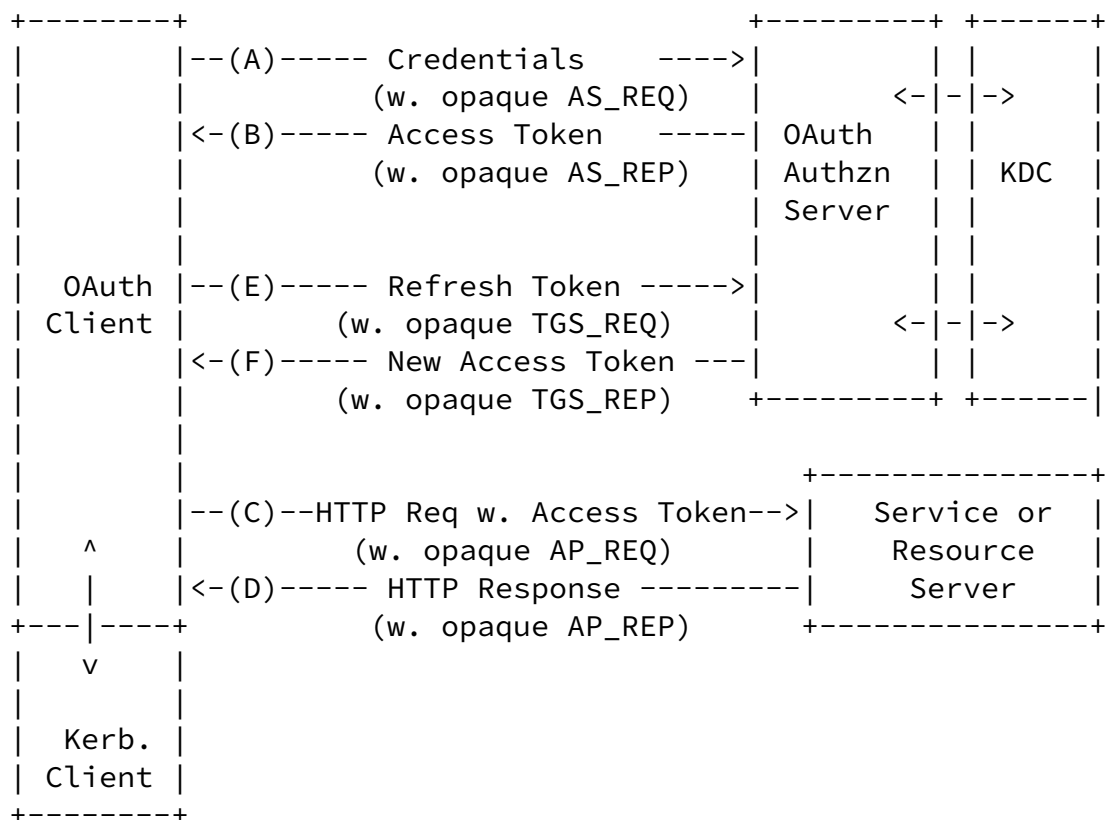


Figure 2

The Kerberos client is assumed to be present in the underlying operating system, within the browser or as part of the web application. Similar to the case of HTTP-Negotiate in which the browser performs a call to the underlying Kerberos client, in this case we assume that the OAuth Client is able to perform a similar call. The details of this are outside the scope of OAuth and of the

current document.

[5.](#) Kerberized Token Exchange Service

TBD

[6.](#) Acknowledgements

TBD

Hardjono

Expires June 11, 2011

[Page 18]

Internet-Draft

Kerberos Support in OAuth 2.0

December 2010

[7.](#) IANA Considerations

TBD

[8.](#) Security Considerations

TBD

All drafts are required to have a security considerations section.
See [RFC 3552](#) [[RFC3552](#)] for a guide.

[9.](#) References

[9.1.](#) Normative References

[Needham-Schoeder]

Needham, Roger. and Michael. Schoeder, "Using encryption for authentication in large networks of computers, Communications of the ACM 21 (12)", 1978.

[RFC4120]

Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", [RFC 4120](#), July 2005.

- [RFC4559] Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", [RFC 4559](#), June 2006.

[9.2.](#) Informative References

- [MIT-Athena]
Steiner, J., Neuman, B., and J. Schiller, "Kerberos: An Authentication Service for Open Network Systems. In Proceedings of the Winter 1988 Usenix Conference. February.", 1988.
- [RFC1510] Kohl, J. and B. Neuman, "The Kerberos Network Authentication Service (V5)", [RFC 1510](#), September 1993.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", [BCP 72](#), [RFC 3552](#), July 2003.

Hardjono

Expires June 11, 2011

[Page 19]

Internet-Draft

Kerberos Support in OAuth 2.0

December 2010

- [WSS-Kerberos]
Oasis-Open, "Web Services Security: Kerberos Token Profile 1.1", 2006, <<http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf>>.

[Appendix A.](#) Document History

Changes from -00

- o Removed OAuth-Kerberos message Flows section due to the huge change occurring between OAuth-draft-5 and OAuth-draft-11. Awaiting stable version of Auth 2.0 spec.

Author's Address

Thomas Hardjono (editor)

MIT Kerberos Consortium
77 Massachusetts Avenue W92-152
Cambridge, MA 02139
US

Email: hardjono@mit.edu