OK: OAuth 2.0 interface for the Kerberos V5 Authentication Protocol

James Max Kanter
kanter@mit.edu

Bennett Cyphers
bcyphers@mit.edu

Bruno Faviero
bfaviero@mit.edu

1. Problem

Kerberos is a powerful, convenient framework for user authentication, and over 60% of medium to large enterprises today use some form of the protocol [3]. OAuth 2.0 is another open source protocol with wide use on the web today, although its goals are slightly different:

The motivation for this project came from a personal need for easily authenticating MIT students when building web apps, especially ones not hosted at MIT. In addition, there are several reasons why an OAuth interface to the Kerberos protocol would be useful to the broader web development community. First, although Kerberos username-password authentication can be used directly in third-party web apps, it requires the user to enter their password on an untrusted web page. Although the password can be processed client-side, this doesnobey the principle of least privilege: ithard for a layperson to verify that their password will not be stored or sent to a remote server when they type it into an unfamiliar site, and they shouldhave to. Encouraging this design pattern only trains users to be careless with their credentials.

Touchstone, based off the Internet2 Shibboleth single sign-on package, sseems designed to address this issue. However, it is somewhat complex to implement [4] and many developers are already familiar with OAuth. Furthermore, OAuth is already used by some of the most popular sites on the Internet, including Google, Facebook, Github, and Twitter. Providing an interface to MIT's database of users (or that of any Kerberos system) would allow developers to add Kerberos authentication by reusing code from their existing application. By using just one protocol for authentication, developers can write less code, deal with fewer edge cases, and spend more time SHIPPING.

2. Design

The OAuth 2.0 specification does not specify how the authorization server should authenticate the user. This means it is possible to incorporate Kerberos as the authentication method.

We want the use of OAuth in place of Kerberos to be transparent to the end user. This mean keeping components using one or another strictly separate and using HTTPS to communicate between the components. The user should be able to grant access to a site for specific Kerberos services, without ever granting the site direct access to either his Kerberos secret or a general ticket-granting ticket on his behalf. Furthermore, the user should only need to authorize the third-party site once per session, and the site should be able to access the services for which it is authorized as many times as it needs until the user's Kerberos session expires.

Unlike the proposal written by the Kerberos Constortium, our system will not assume that the OAuth client also has access to a Kerberos Client. This means the user must securely transmit their password to the OK server. However, the server never stores this password. For security purpose, the server to be stateless. All information needed to handle requests will be saved in access token rather than stored more permentently on the server.

We will follow the diagram in figure 1.

2.1. Authentication

Before a OAuth client can communicate with the OK server it must register in order to obtain a secret key to authenticate authentication requests.

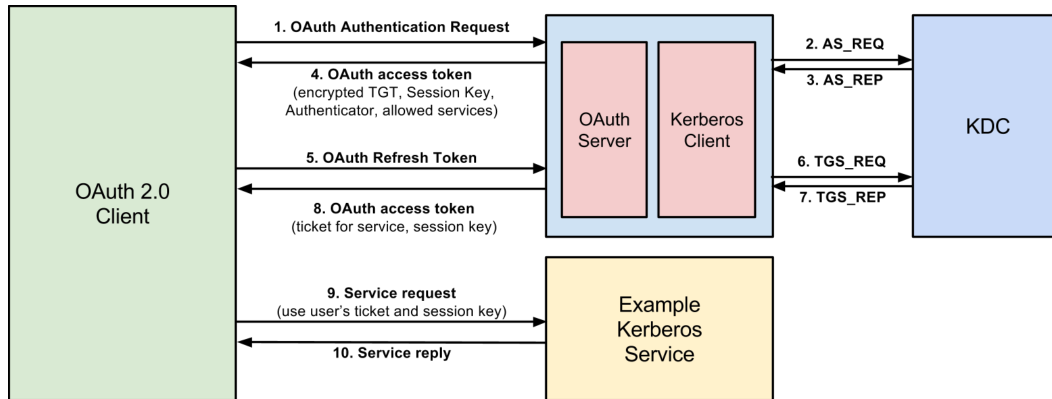

Figure 2. Process for authenticating, requesting service access, and accessing a service

When a user tries to log in to a site using the "OAuth-Kerberos" login option, the website redirects the user to the OAuth-Kerberos server for authentication (step 1.). On the OAuth-Kerberos site, the user passes their password over HTTPS and the OK server send AS_REQ with this information to ask the MIT KDC for a ticket-granting ticket (step 2.). The kerberos client can decrypt AS_REP using the client's password (step 3.).

The OAuth-Kerberos server returns a encrypted access token that contains the TGT, the authenticator, and a list of the services for which the user has granted access (step 4.). This token can only be decrypted by the OK Server. By including the authenticator, this access token inherits the same lifetime as it would in a normal Kerberos session (on the order of minutes).

2.2. Service request

When that site would like to act on the user's behalf to a Kerberos service, the client sends the refresh token request to the OK server along with the service that it would like to access and the original access token (step 5.).

The OK server gets this information, decrypts it, and sends TGS_REQ to request a ticket from the Ticket Granting Server (step 6.). The TGS returns TGS_REP, which is a service ticket and session key (step 7.). This information is sent back to the OAuth client to use to connect to the service (step 8.)

2.3. Service access

The site now has everything it needs to access the desired service: a ticket for that service, and a session key with which it can encrypt an authenticator and decrypt the service's replies. The exact details of communicating with a service are dependent on that service implmentations. Step 9 and 10 represent this process.

3. Implementation

We will implement the following entities:

- An example website that knows how to obtain keys for different services and communicate with those services by sending a ticket, authenticator, and session key.
- An example service that is able to communicate with a client based on the above pieces of information.

We will take advatange of existing libraries to implement the OAuth 2.0 and Kerberos protocol, including:

- Python OAuth 2 library [2]
- Python GSSAPI to communicate with Kerberos: [1]

References

[1] Python-gssapi. http://pythonhosted.org//python-gssapi/.

[2] Python oauth 2.0 library. https://github.com/simplegeo/python-oauth2.

[3] Thomas Hardjono. Oauth 2.0 support for the kerberos v5 authentication protocol, December 2010.

[4] Paul B. Hill. Provisioning steps, June 2014.