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
John Peebles jpeebles@mit.edu

1. Problem

Kerberos is a powerful, convenient framework for user authentication and authorization. Within MIT, Kerberos is used with many online institute services to verify users as part of Project Athena. However, it can be difficult for developers unfamiliar with Kerberos development to take advantage of its resources for use in third-party apps.

OAuth 2.0 is an open source protocol used across the web for secure delegated access to resources on a server. Designed to be developer-friendly, OAuth is the de facto standard for authenticating users across sites, and is used by services including Google, Facebook, and Twitter.

Our goal with OK Server is to provide an easy way for developers to access third-party services using Kerberos via OAuth. The benefits of this are twofold: developers can rely on an external service for user identification and verification, and users only have to trust a single centralized server with their credentials. Additionally, developers can request access to a subset of Kerberos services on behalf of a user.

2. Implementation overview

Our system is composed of two main components: a server (the Kerberos client) to retrieve and process tickets from the MIT KDC, and an OAuth interface (the OAuth server) to interact with a client app (the app) wishing to make use of Kerberos authentication. When using our system, a client application uses the OAuth protocol to get Kerberos service tickets for a particular user.

The code can be found at `https://github.com/bfaviero/ok`.

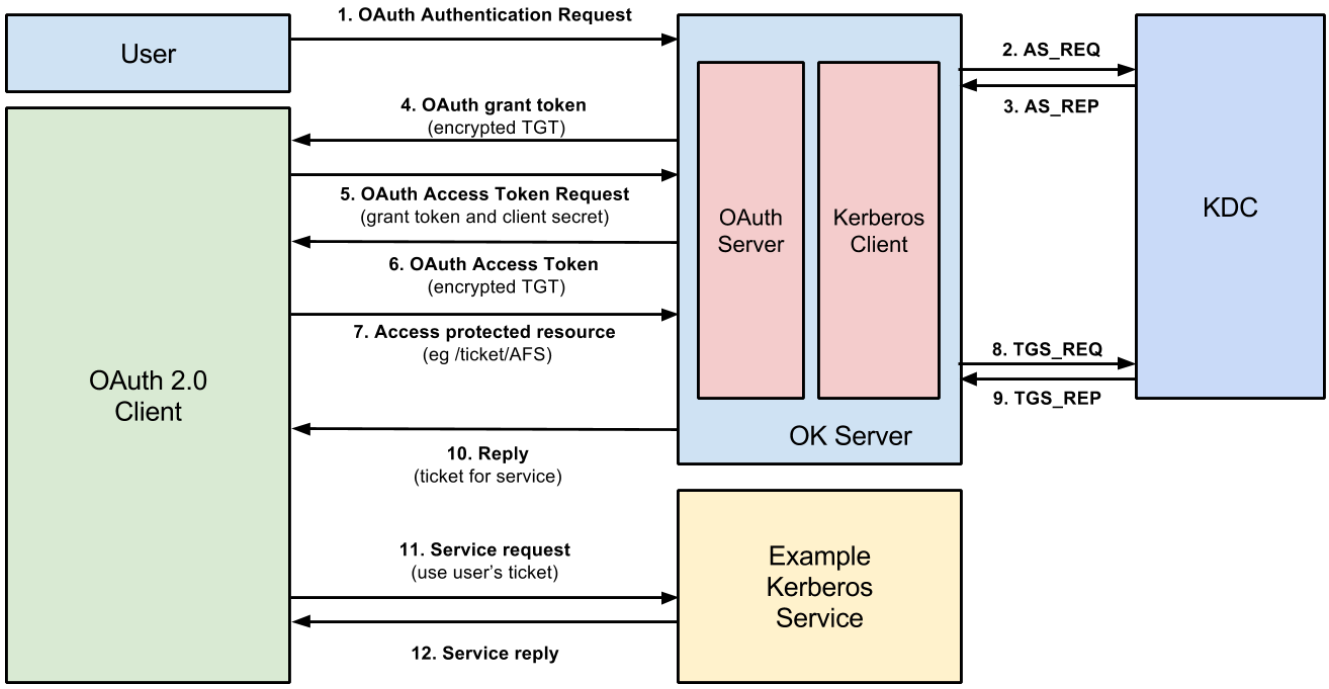
3. Goals

We designed OK Server with the following goals in mind:

- Familiar API for developers
- Familiar interface for users
- Granular permissions control for users
- Security, particularly when handling Kerberos credentials and tickets

4. OAuth-Kerberos full workflow

The full description of how our system works is shown in the diagram below.



The client is first registered with the OAuth server, and lists which services its users should be allowed to access.

1. A user visits the OAuth clients website and is redirected to OK Server, where the user enters his or her credentials and authorizes the client to access the listed services on behalf of the user.

2–3. The OK Server uses these credentials to fetch a ticket-granting ticket (TGT) from the realm Key Distribution Center (KDC). The TGT is serialized and encrypted in the OK Server's private key.

4. OAuth Server returns a grant token to the OAuth client. This token contains the encrypted TGT and expiration date. This grant token cannot be used yet to acquire services tickets on behalf of the user.

5. The OAuth Client requests an access token by sending the grant token along with their client secret.

6. The OAuth Server verifies that the the user has approved the client, and that the client is in fact who they say they are with the client secret. The server sends the access token, which also includes the encrypted TGT and expiration date, to the client.

7. The client acquires a service ticket by accessing an OAuth protected resource on OK Server. Because the service is protected by OAuth, the OAuth client must include an access token, which contains the TGT.

8–9. The server decrypts the TGT from the step 7 and uses it to request a service ticket from the realm Ticket Granting Server.

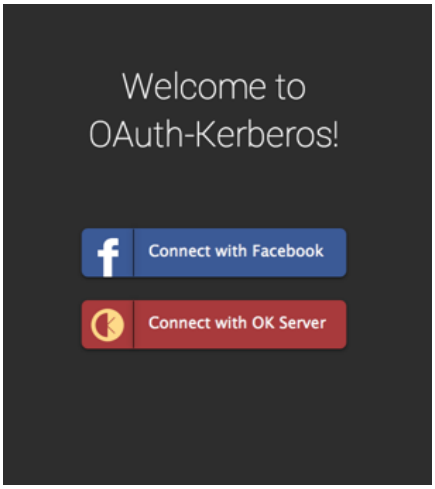
10. The OK server returns the service ticket.

11–12. The OAuth Client now has temporary access to the service on behalf of the user, and uses the service ticket as they normally would until it expires.

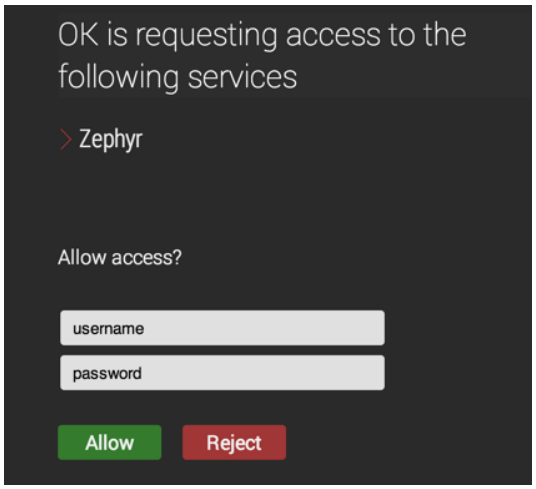
Note: The server only stores the registered clients and their metadata throughout the entire process. In particular, it is not necessary for the server to store any passwords or tickets between requests.

5. User experience

The flow to login to a site that uses OK Server is the same as what users experience on many other websites.



(a) The login screen



(b) OAuth server lists the services that the client will have access to before authorization

6. Implementation challenges and security considerations

6.1. Client registration

When a client application is registered, the developer must specify at the time of registration which services the client may ask to access on behalf of the user. The client is issued a secret key that it must use when it wants to interact with the server. Utilizing this secret key-authentication method allows OK Server to revoke access from clients that are abusive or have been compromised.

6.2. Kerberos Client

To interact with Kerberos at a sufficiently low level, we extended a python library (in the lib/ folder of our project) written by David Benjamin [3] to wrap Kerberos functions from the krb5 C library. In kerberos_client.py, we define a set of functions to carry out the necessary actions:

- `get_tgt()` takes a principal and password, and returns a serialized ticket-granting ticket.
- `get_service_ticket()` takes the serialized TGT and returns a serialized service ticket for a specified service.
- `store_service_ticket()` takes a serialized ticket and stores it in a temporary credentials cache for use by other applications, such as openafs. This is for use by client applications which have used OAuth to access the service.
- `clear_service_ticket()` removes the stored service ticket from credentials cache.

To serialize the TGT and service ticket, we used the built-in serialization functions for tickets provided in libkrb5, `krb5_mk_1cred` and `krb5_rd_cred`, which create and process kerberos credentials as `krb5_data` structures. The python wrappers, `serialize_cred` and `deserialize_cred`, are defined in kerberos_serializer.py.

6.3. Encryption

When passing around tickets over the network, OK Server enforces the use of HTTPS, so traffic can't be compromised by man-in-the-middle attacks. However, it is also important that the client is unable to retrieve the actual TGT from the OAuth access token: otherwise, it could could access any service available to the user. To this end, OK Server encrypts all tokens that contain a TGT using the OAuth server's secret key. We use AES with Offset Codebook Mode (OCB). OCB mode provides built-in integrity checking with the cipher, so the system does not have to worry about the possibility of tampering.

6.4. Trust

The main security consideration is that the end user has to trust the OK Server with the password. This is in contrast with most Kerberos implementations, where the user's password never leaves the machine. This means that the user must trust OK server, and it's critical that the server not be compromised. In our implementation, the user's password is passed over an HTTPS connection, and only stored long enough to be able to fetch a TGT. Therefore, in the event that OK server is compromised, anyone who has authenticated in the past is safe.

Furthermore, we emphasize that the user does not have to trust the client with their password or TGT. Any application that accesses a service on a user's behalf never learns the user's password or has direct access to the TGT at any time.

7. Related work

We did not find any implementations an OAuth server backed by Kerberos authentication. There is an RFC [4] from the Kerberos Consortium which describes a service roughly similar to the one we built, but our design varies in that it does not require a client app to have an implementation of Kerberos.

Webathena [3] is a client-side Kerberos-interaction system that stores a user's tickets in the browser, and then uses a server proxy to make use of those tickets to authenticate to specific resources. We built upon Webathena's Python libkrb5 wrapper [2] to interact with the Kerberos protocol, and made significant additions to it.

The library we relied on most is libkrb5 [1], the official reference library, written in C, for interacting with MIT Kerberos. The official documentation is sparse with few examples, and we did not find many useful references on the web either. However, after some trial and error, we were able to take advantage of the powerful, low-level interface for Kerberos which it provides.

8. Contributions

1. Made OAuth API for developers to incorperate Kerberos
2. Developed a Python library for obtaining TGT and service tickets
3. Provided example client that allows users to authenticate MIT students and access a user's AFS files

Overall, OK Server should make it easier for developers to quickly build and deploy applications that take advantage of ubiquitous Kerberos infrastructure without sacrificing user security.

References

[1] Github: krb5. `https://github.com/krb5/krb5`.

[2] Github: Roost-python. `https://github.com/roost-im/roost-python`.

[3] David Benjamin. Adapting kerberos for a browser-based environment. Master's thesis, MIT, 2013.

[4] D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749 (Proposed Standard), October 2012.