Authentication Servers

In a large distributed systems, keeping track of information about all of the authorized users and what each is authorized to do may be a great deal of work. We have discussed Public Key Certificates as a means of authenticating ones' self to a server that was not previously aware of us or able to engage us in a password dialog. But ...

- it may not be practical to require all actors to have registered public/private key pairs, or for all servers to use them as the basis for authentication.
- even if we have an authenticated identity for an actor, we still may need to be able to access and interpret a large access control database to determine whether or not they are allowed to perform a particular operation.

At some point, it becomes unreasonable to expect application level services to be able to decide who should be allowed to do what. And we ask if Authentication and Authorization checking can be turned into services.

Authentication and Authorization as a Service

We introduced the concept of Certificate Authorities, as trusted agents who we trusted to authenticate Public Key Certificates ... and, by implication, the identity of the owner of the associated private key. This concept of delegated trust can be applied to a wider range of authentication and authorization decisions.

A public key certificate is one type of *credential*, which we trust because it was signed by someone we trust. We could use a similar technique to get trusted *capabilities*:

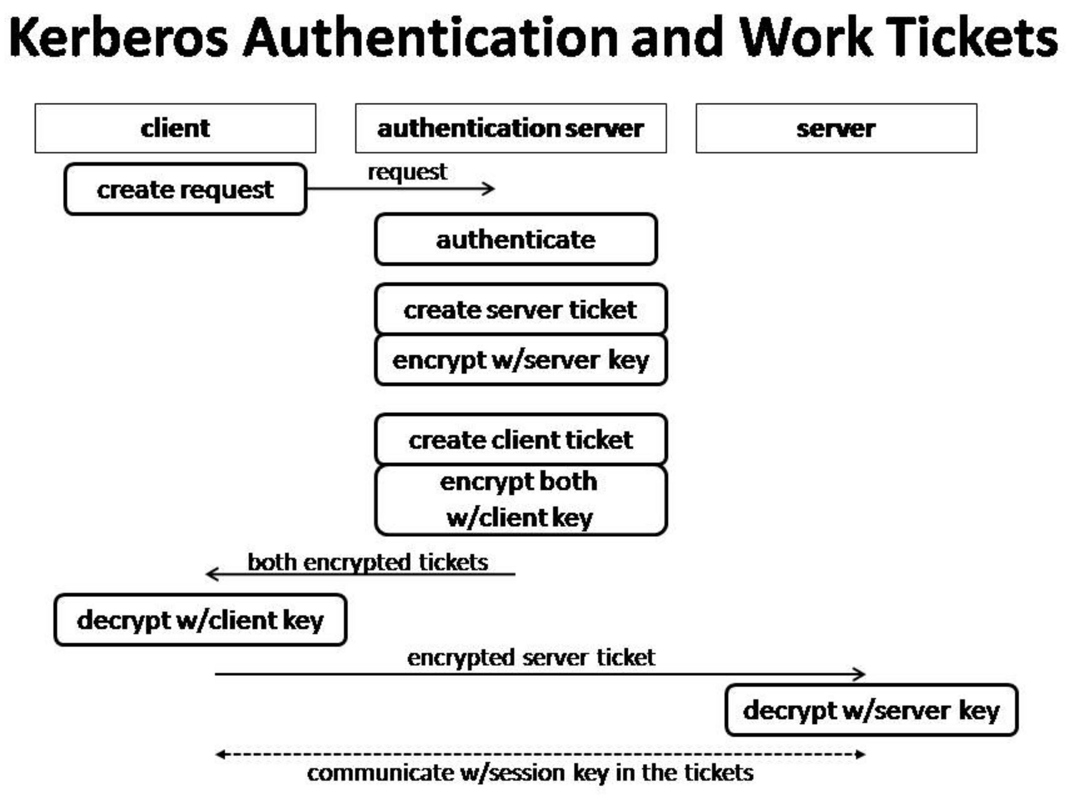
- an actor would contact an authentication server, and describe the actions to be performed.
- the authentication server would both authenticate the actor and determine whether or not that actor was allowed to do the proposed operations.
- after a successful access check, the authentication server would create a *work ticket* describing the actor, the permitted operation, and any other relevent information (e.g. "good until date"), and then sign it (e.g. with his private key)
- the actor would present the *work ticket* to the appropriate server along with the request.
- the server would not have to do any authentication or authorization checking beyond verifying the validity of the work ticket.

Such a model relieves application servers of all responsibility for authentication and authorization checking. It also completely isolates them from the particular mechanisms and policies by which identities are authenticated and privileges ascertained.

Work Tickets

The above described *work tickets* are unforgeable, in that the signature prevents random agents from creating work tickets. If they include expiration dates, they cannot be reused. But, as described, they are transferrable (or stealable). A server would honor any request described by a valid ticket. How can we ensure that the ticket can only be used by the authorized agent/actor?

We could certainly include the public key of the authorized agent in the ticket, but this would get the server back into the authentication business. Another common approach is to have the authentication server generate a session key, and encrypt a copy for the client with the client's public key, and a copy for the server with the server's public key. In this way, each party can be confident that it is talking to the authorized agent.



Each work ticket contains a session key, generated by the authentication server. When the server receives the work ticket, it can verify the authentication server's signature and so trust that the client has been authenticated. Because the server ticket was encrypted with the server's key, the client can be sure that the agent at the other end of the connection is the requested server; nobody else could decrypt the session key in the server ticket. Because the server ticket had also been encrypted with the client's key, the server can be sure that the agent at the other end of the connection is the authenticated client; nobody else could have decrypted and forwarded the server ticket.

Work Tickets without Public Key Encryption

The above has been described in terms of public key encryption, but similar processes can be implemented with symmetric encryption. In Kerberos, each actor had a symmetric key which was shared only with the authentication server. These keys could be used:

a. to authenticate a ticket as having come from the authentication server
b. to communicate information which can only be read by actors of the authorized agent

Summary

An authentication service can completely eliminate the need for application servers to know about authorized actors and how to authenticate them. Work tickets are examples of unforgeable capabilities that can eliminate the need for applications to make authorization decisions. Even in closed systems (where authentication and authorization checking are not issues), work tickets are a common representation for granted leases.