Stateless authentication

The stateless authentication is achieved by returning two signed json web tokens from isis. The calling client is responsible for maintaining its tokens.

The refresh token contains the following information:

- The user name
- An expiration date 15 hours after log in. (This value is configurable under the parameter "isis.token.refresh.ttl")
- A signature based on isis' secret.

The refresh token is returned in a cookie. Once the expiration date for the refresh token has passed, the user will have to be asked to provide user identifier and password again to acquire a new refresh token. The refresh token can be used to acquire an access token.

The access token contains the following information:

- The user name
- An expiration date 20 minutes after log in. (This value is configurable under the parameter "isis.token.access.ttl")
- All of the permissions for the user.
- A signature based on isis' secret.

The access token is presented to other services in the system for authentication and authorization. To authenticate access to a service, the signature on the access token is checked against the isis signature with which the service was initialized for a given tenant. To authorize access to a service, the access token is checked against a list of endpoints in that service which have been annotated as @Permittable(AcceptedTokenType.TENANT). The saving of the isis public key in initialization, the authentication, and the authorization are all performed by anubis. The microservice only needs to annotate its endpoints appropriately.

Once the expiration date for the access token has passed, the client must present the refresh token via cookie for a new access token. The most current role/permissions for the user will be used to assemble the access token. Hence role/permission changes will take a maximum of 20 minutes to take effect.

The algorithm we use for signing is RSA. We decided against ECDSA which is slower than RSA, but produces smaller tokens. We chose an asymmetric algorithm (rather than a symmetric one like HMAC), so that we can distribute the public key easily to the services which work together without endangering the security of the system. We chose to hardcode the algorithm in isis and anubis to prevent attacks in which the algorithm in the token is adjusted by the attacker (for example to "none"). The secret is allocated per tenant, so that users cannot be authenticated for any tenant other than the one in which their account is managed.

Services using the access token for authorization, will need to run on systems with clocks with the correct time.

Reference:

- https://tools.ietf.org/html/rfc6749
- https://auth0.com/blog/2015/03/31/critical-vulnerabilities-in-json-web-token-libraries/
- https://auth0.com/blog/2015/12/17/json-web-token-signing-algorithms-overview/