Proposal: JWT/OAuth/OIDC Authorizer

Problem Statement

Today Cadence has an AuthN interface and already plumbed through the system. However, it currently only has a noop implementation in the open source repo. To enable authorization, Cadence requires users to implement the interface and then re-compile the binary to use. This is not convenient because users have to be familiar with Cadence code base in order to make the code changes. Moreover, they will not be able to use the official docker image from Cadence releases, and have to keep their code in sync with Cadence, which will be very difficult to do for most users.

Usually different companies have their security systems for authorization, and vary in different solutions. Here we want to provide Cadence users an easy and unified way to integrate with their security solution, without writing/changing any code in Cadence code base, so that users can use the official released images.

OpenID Connect(OIDC) is a commonly adopted protocol to authenticate/authorize users in WebUI. It is a simple identity layer built on top of the OAuth 2.0 protocol, which allows clients to verify the identity of an end user based on the authentication performed by an authorization server or identity provider (IdP), as well as to obtain basic profile information about the end user in an interoperable and REST-like manner. OpenID Connect specifies a RESTful HTTP API, using JSON as a data format.

Proposal

Previously we proposed Cadence to provide a configurable Webhook Authorizer via HTTPS to an authorized external service. In that proposal Cadence will rely on the external service to determine if a request has enough permission. However this approach requires users to implement a web service. It's not efficient, and not easy to implement.

We got excellent feedback from the community that we can use the OAuth protocol based on JSON Web Token(JWT). This JWT Authorizer will implement the current AuthN interface using the OAuth protocol. But it's not completely the same because Cadence is running on TChannel(RAW TCP) or gRPC instead of HTTP/HTTPS. Based on that, Cadence Web will support OIDC for integration. Especially in many companies, OIDC are extended with "groups" scope to identify users' permission groups. For example, OneLogin. The "groups" scope is a perfect use case to manage the permission for different teams/service in a company.

Note that when this document is being written, gRPC support is still in process in Cadence project. Cadence will be deprecating TChannel in the future. Therefore this proposal suggests to implement the authorization for TChannel at beginning and then gRPC later. gRPC is based on gRPC metadata.

This proposal will use community JWT library for implementation. Only support RSA 256 as the initial version.

Cadence Server Configuration

Below Authorization will be a member under Config.

```
Authorization struct {
  OAuthAuthorizer OAuthAuthorizer `yaml:"oauthAuthorizer"`
  NoopAuthorizer NoopAuthorizer `yaml:"noopAuthorizer"`
}

NoopAuthorizer  struct {
  Enable bool `yaml:"enable"`
}

OAuthAuthorizer struct {
  Enable bool `yaml:"enable"`
  // Credentials to verify/create the JWT
  JwtCredentials JwtCredentials `yaml:"jwtCredentials"`
  // Max of TTL in the claim
  MaxJwtTTL duration `yaml:"maxJwtTTL"`
}

JwtCredentials  struct {
  // suuport: RS256 (RSA using SHA256)
  Algorithm string `yaml:"algorithm"`
  // for verifying JWT token passed in from external clients
  PublicKey string `yaml:"publicKey"`
  // for creating JWT token
  PrivateKey string `yaml:"privateKey"`
}
```

JWT Format And Example

This is the JWT format that Cadence server will be used to authorize requests. Note that this is not the same as the JWT of OIDC protocol. This is because Cadence server is not only providing service to user experience, but also for workflow applications.

Cadence Web will integrate with OIDC and then generate a new JWT with this format. See “Cadence Web implementation with OIDC” section for more details.

JWT Claim Field	Explaination
name	Needed for WebUI to show who is current user
admin	bool to indicate that a user is admin. If true, then “groups”

	can be omitted
groups	The permission groups that users has associated with
iat	issued at time
ttl	TTL: How long a JWT can be valid since timeAtIssue(iat)

Example:

```
{
  "name": "John Doe",
  "admin": false,
  "groups": "group1@indeed.com",
  "iat": 1516239022,
  "ttl": 86400,
}
```

Authorizer(Server)

Permission Storage

Cadence domain data will be used to store all permission information. It's a free-form KV for generic purposes. This proposal is going to preserve two keys:

- **READ_GROUPS**: the permission groups that have read permission
- **WRITE_GROUPS** : the permission groups that have write permission

NOTE There are three levels of permission:

- **read**: means readOnly on the domain level APIs
- **write**: means read+write on domain level APIs
- **admin**: means read+write on any domains, and required for cluster level APIs like registering domains

This permission storage can only be updated by Cluster admin.

For examples, it can be maintained by CLI:

```
./cadence --do samples-domain domain update --domain_data "READ_GROUPS=group1@indeed.com
group2@indeed.com, WRITE_GROUPS=group3@indeed.com group4@indeed.com serviceA serviceB"
Domain samples-domain successfully updated.
```

Then it's available like below

```
./cadence --do samples-domain domain desc
Name: samples-domain
```

```
UUID: cfebaf91-d58d-46fe-9951-345c357332bf
Description:
OwnerEmail:
DomainData: map[ WRITE_GROUPS:group3@indeed.com group4@indeed.com serviceA serviceB
READ_GROUPS:group1@indeed.com group2@indeed.com]
...
...
```

NOTE: serviceA/serviceB is an example for Cadence client access. Normally it's always requiring write access as polling and responding workflow/activity tasks are write requests.

When onboarding a new domain, or updating the group permission, cluster admin can use CLI tool to update the domain data.

Implementaiton

OAuth authorizer will be implemented as an interceptor using the configuration above. The approach is similar to this blob.

The authorization will be done as follow:

- Verify the JWT token using **JwtCredentials** and **JwtTTL**
- Authorize using the claims fields **admin, groups** and **domain** and the APIs' read/write required that being hit
    - Pass if it's admin, otherwise next step
    - Get the domain data based on the domain name
    - Based on the read/write required by the API, get the list of associated groups by reading "READ_GROUPS" or "WRITE_GROUPS"
    - Verify the groups

Authorization Provider(Clients)

Internal clients talking to frontend

There are internal components like batcher/parentClosePolicy/XDC/archival/etc that are talking to frontend services. We have to authorize the requests.

The JWT will be created using **JwtCredentials** configuration. This will be easily implemented after Go Client has a default implementation of "AuthorizationProvider" using a private key.

Web

OIDC with group information

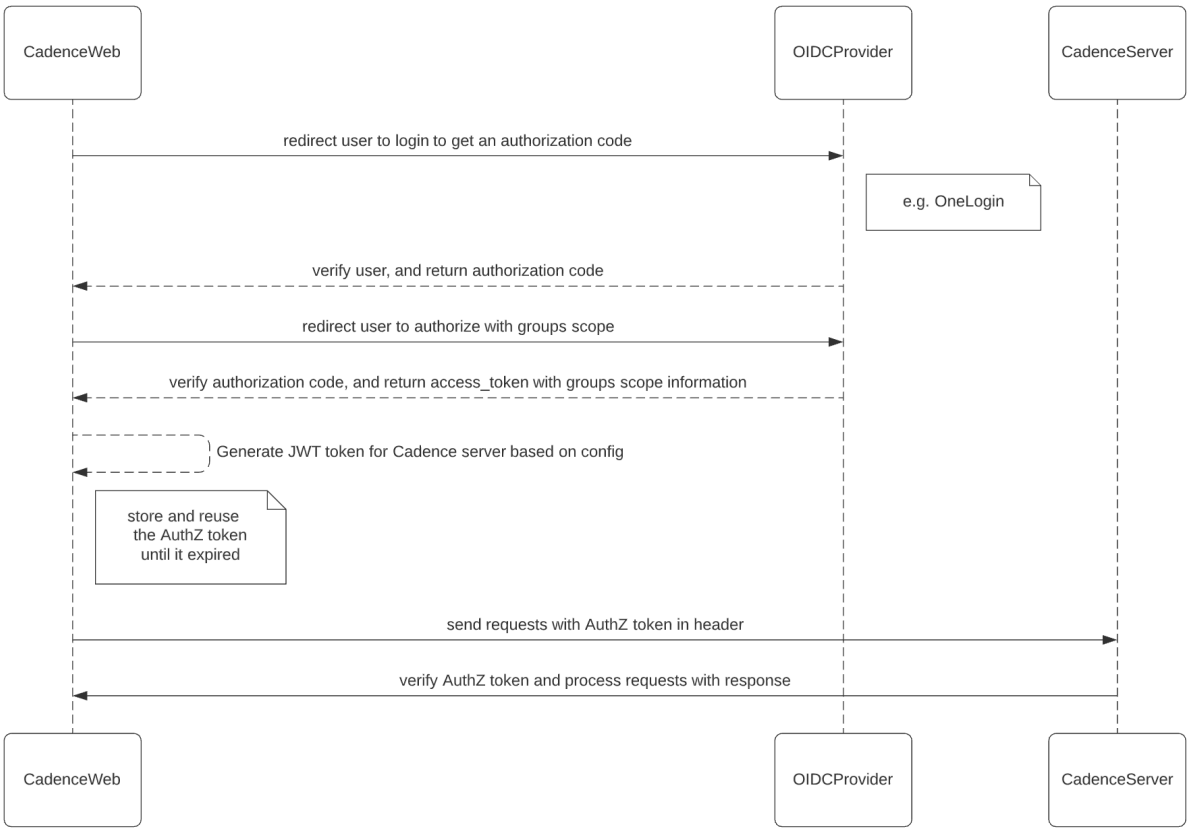
It's recommended to use nn OIDC provider which supports "groups" scope to integrate. An example is OneLogin's OIDC. Without "groups" scope, all users will have to be granted "admin" permission.

Example configuration in Web UI:

```
authorization
  oidc:
    clientId: your-onelogin-oidc-app-client-id
    clientSecret: your-onelogin-oidc-app-client-secret
    accessTokenUri: https://<subdomain>.onelogin.com/oidc/2/token
    authorizationUri: https://<subdomain>.onelogin.com/oidc/2/auth
    tokenName: id_token
    authorizedGrantTypes: authorization_code
    scope: email,groups
  jwtCredentials:
    privateKey: <path to privateKey file to generate JWT token>
    ttlSeconds: 86400
    adminGroups: [group1, group2]
```

Implement OIDC protocol:

- Ask user to login if doing any operation:
  - Not yet logged in, Or
  - The previous has expired
- Have a button to let a user login. The login URL will kick off OIDC flow:
  - Redirect the user to accessTokenUri to get an authorization code,
  - Then redirect the user to authorizationUri to get the access token by "tokenName"(in OneLogin, this is "id_token").
  - Extract "groups" information from access token to generate JWT using server's JWT format.
    - If one of the groups from OIDC is in "adminGroups" in config, then generate it with admin permission
  - Store the generated JWT token into local storage
- Use the generated JWT token from local storage to talk to Cadence server if not expired
- Have a button to export the JWT token into ENV variable for CLI usage(like AWS CLI)
  - export CADENCE_CLI_JWT=xxxxxxxxxx
- Have a button to let users to logout, the information should be deleted from local storage



OIDC without group information

The group information is not a standard part of OIDC so some OIDC providers don't have it. For example, Google OIDC.

In such cases, the OIDC implementation will have to just grant admin access to all logged in users.

Without OIDC

In some cases, users may not have OIDC but they have used other mechanisms to authorize the Web already. For example, they could use a web proxy on nginx server.

For such uses, Cadence Web allows configuring a private key to send authorized requests with admin access.

CLI

- CLI will be allowed to pass a JWT directly or via environment variable:

```
./cadence --do test-domain --jwt <> domain desc
```

Users will get this JWT from the Web UI. See “Cadence Web Implementation with OIDC” section.

- Use JWT credentials of RS256 to create JWT. It will always generate a JWT with “admin:true” and omit the “group” field.

```
./cadence --do test-domain --jwt-alg RS256 --jwt-private-key <> domain
desc
```

- CLI can configure with a webhook to redirect users to a page to login, and then the login will redirect back to provide the JWT

Client SDK

Create an interface as AuthorizationProvider.

In java:

```
public interface AuthorizationProvider{
   // getAuthToken provides the OAuth authorization token
   // It's called before every request to Cadence server, and sets the
token in the request header.
   []byte getAuthToken();
}
```

In Golang:

```
interface AuthorizationProvider{
    // GetAuthToken provides the OAuth access token...
    GetAuthToken() byte[]
}
```

One of the implementations is to allow configuring the Private keys to sign/create JWT locally.

The interface is exposed for users to get valid JWT in a customized way, e.g. if they need to talk to an external service.

Future work to consider

- Secret key rotation
- More algorithm support
- Different secret keys for different domain

Task(PR breakdown)

Task	Note	Prioirity
server authorizer with unit test	PR1 PR2 PR3	1
CLI authorized requests by env and JWT private key	PR1 PR2	1
Golang client: AuthProvider interface and workerOption	PR1	2
Golang client: implementation of AuthProvider interface with private key	PR1	2
Web sending authorized requests with private key	PR1	3
Web OIDC integration and sending authorized requests		2
Internal components(batcher/parentClosePolicy/etc in worker service) talking to Frontend	PR1	2
Java client: AuthProvider interface and workerOption	PR1	2.5
Java client: implementation of AuthProvider interface with private key	PR1	3
Server: use authorizer in AdminHandler	PR1	3
Provide better error message if JWT is already expired		3
Docker compose file	PR1	4
Server integ tests to protect the feature		4
CLI authorized requests by URL hook and callback		3
[Improvement]Load credentials on startup	PR1	3
Go client: provider option to use AuthProvider for raw service client. In some cases Client/DomainClient are not enough		3.5

Alternatives considered

- Proposal: OIDC(OpenID Connect) + ApiKey DualAuthorizer
    - Pros
        - No code needed at Indeed to Integrate
        - Token can invalidated earlier when refresh

    - Cons
        - Cannot customize service authN/Z, e.g. Indeed's internal service auth
        - The implementation in Cadence will be more complex
            - CLI and Web will need refresh tokens periodically
            - CLI will integrate with with OIDC directly and cannot reuse Web's token
        - Cadence server need to verify the JWT token from OIDC
            - Call OIDC(oneLogin) will not be a good idea for availability/latency
            - Not sure if we can get the public key from OIDC(OneLogin)
- Proposal: Configurable Webhook Authorizer
    - Pros:
        - The implementation on Cadence will be very simple
    - Cons:
        - Cadence will have a dependency on the auth service on each request
        - The design/implementation of integration at Indeed will be complex

Open Questions

-