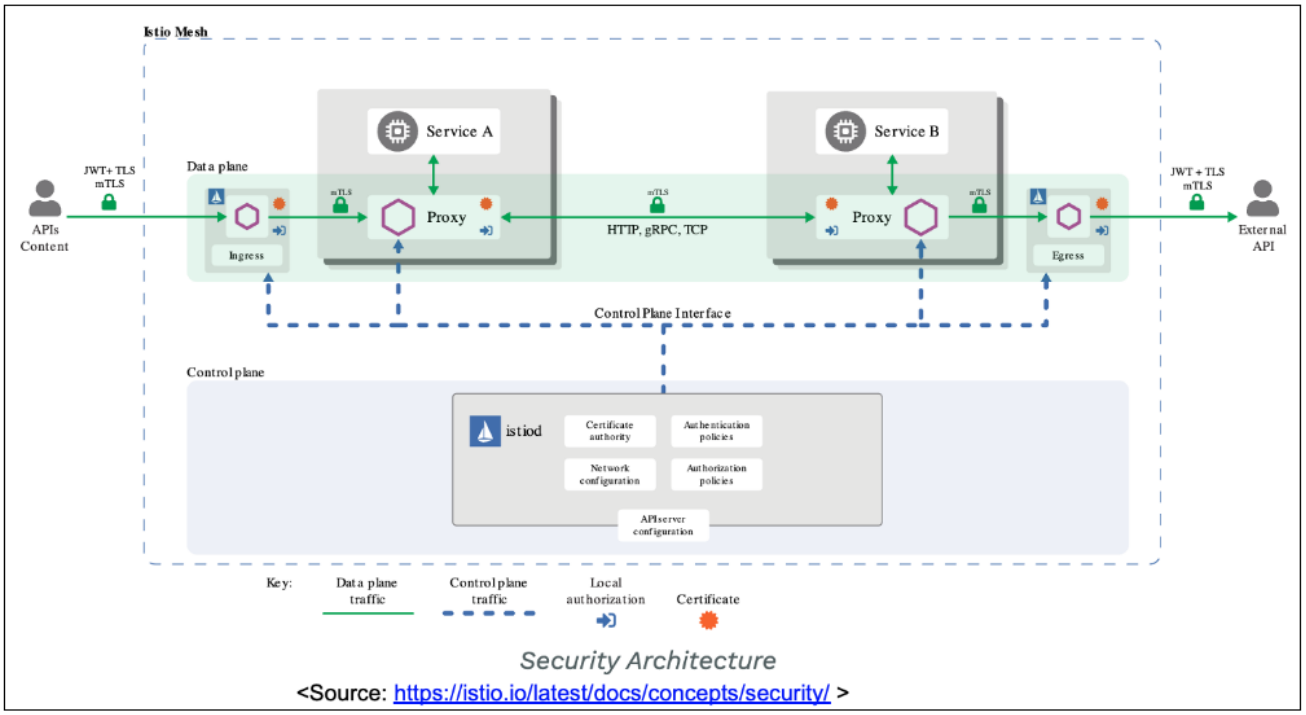
AuthN + AuthZ

Chapter 1: How ONAP Did It in London

Please see ONAP Next Generation Security & Logging Architecture.



<Source: https://istio.io/latest/docs/concepts/security/ >

London release of ONAP is shipped with an authentication solution based on this article It provides basic authentication capabilities but lacks access management on a per-user basis.

How it Works

- You need to have OAuth2-Proxy deployed:

```
kubectl get pods | grep oauth2
```

Example output:

```
onap-platform-onap-oauth2-proxy-8c5cbcf57-5vc4l 2/2 Running 0 43h
```

- You need to have Keycloak deployed:

```
kubectl get pods -n keycloak | grep keycloak
```

Example output:

```
keycloak-0 2/2 Running 0 2d23h
keycloak-db-postgresql-0 2/2 Running 0 2d23h
```

- Keycloak needs to have OAuth2-Proxy configured as "client," and OAuth2-Proxy must have Keycloak configured as Identity Provider.
- Istio must have OAuth2-Proxy configured as "provider" in the global Istio config:

```
kubectl edit cm istio -n istio-system
```

All of the above comes from ONAP after deployment.

Now, to make it work, apply additional `AuthorizationPolicy` :

```
apiVersion: security.istio.io/v1
kind: AuthorizationPolicy
metadata:
  name: echo-authz
  namespace: istio-ingress
  resourceVersion: "31013"
  uid: e8238e5b-08f9-48ab-9247-a57a64f4f5b9
spec:
  action: CUSTOM
  provider:
    name: oauth2-proxy
  rules:
    - to:
      - operation:
          hosts:
          - echoserver.simpledemo.onap.org
    selector:
      matchLabels:
        istio: ingress
---
```

Key understanding here is that OAuth2-Proxy does not really work as a real proxy...

Notes

Community used an experimental configuration format for OAuth2-Proxy aka "alpha configuration," meaning some options are not implemented or failing to work.

Chapter 2: How Can We Fix/Extend What the Community Did?

Option 1: Extend Base Scenario with Group Support

This option allows to use User Groups present in Keycloak. That way at least not all users in Keycloak (or other Identity Provider) will have access to the platform, this can be done:

- 1a. by adding to oauth2-proxy option, "allowedGroups", that should do the trick, community for some reason has commented out this part (not sure why), I have not tested it (just noticed it lately). The disadvantage of it is that this will work for all components, and per all actions. And we cannot differentiate between readers and admins.
- 1b. by extracting groups from the claim to the header and allowing this header to be used by istio for validation. Then we have much more granularity. How it works?

Edit oauth2-prox2 configuration add following lines:

```
 server:
          [...]
             BindAddress: '0.0.0.0:44180'
           injectResponseHeaders:
          [...]
           - name: x-auth-groups
             values:
             - claim: groups
```

What it does is it extracts groups from JWT token and maps it for header x-auth-groups (any name here is good) Edit main istio config add following lines:

```
mesh: |-
          [...]
          extensionProviders:
          - envoyExtAuthzHttp:
              headersToDownstreamOnDeny:
              [...]
              headersToUpstreamOnAllow:
              [...]
              - x-auth-groups
              includeHeadersInCheck:
              - authorization
```

```
      - cookie

      [...]
```

What it does is it tells istio to relay header from "accept" oaut2-proxy response to destination service.

Create new AuthorizationPolicy:

```
apiVersion: security.istio.io/v1
kind: AuthorizationPolicy
metadata:
 [...]
  namespace: onap
  [...]
spec:
  action: ALLOW
  rules:
  - to:
    - operation:
        hosts:
        - echoserver.simpledemo.onap.org
        methods:
        - GET
    when:
    - key: request.headers[x-auth-groups]
      values:
      - '*onap_reader'
  - to:
    - operation:
        hosts:
        - echoserver.simpledemo.onap.org
        methods:
        - GET
        - POST
        - PUT
    when:
    - key: request.headers[x-auth-groups]
      values:
      - admins*
  selector:
    matchLabels:
      app: echoserver
```

What it does it attach new policy to envoy proxy on "echoserver", this policy checks the value of x-auth-groups header and block request that does not match either of rules. Please note that this setup allows as to configure access per user group, per host (service), per operation (can be also extender on per path base) Thats quite ok granularity.

Option 2: Replace Community OAuth2-Proxy Configuration with Non-Experimental Configuration

That allows more operations, especially it allows to forward jwt tokens from ingress further to service mesh. We have the option to use one of the two jwt tokens (identity, or user info).
First, we need to edit oauth2-proxy configuration via editing its deployment:

```
apiVersion: apps/v1
kind: Deployment
[...]
    spec:
      automountServiceAccountToken: true
      containers:
      - args:
        - --provider=oidc
        - --provider-display-name="AmartusKeycloakID"
        - --client-id=oauth2-proxy
        - --client-secret=5YSOkJz99WHv8enDZPknzJuGqVSerELp
        - --oidc-issuer-url=https://keycloak-ui.simpledemo.onap.org/auth/realms/ONAP
        - --oidc-jwks-url=http://keycloak-http.keycloak/auth/realms/ONAP/protocol/openid-conne
        - --profile-url=https://keycloak-ui.simpledemo.onap.org/auth/realms/ONAP/protocol/open
        - --validate-url=https://keycloak-ui.simpledemo.onap.org/auth/realms/ONAP/protocol/ope
        - --redeem-url=http://keycloak-http.keycloak/auth/realms/ONAP/protocol/openid-connect/
        - --scope=openid email profile groups roles
        - --skip-oidc-discovery=true
        - --cookie-secure=false
        - --cookie-secret=CbgXFXDJ16laaCfChtFBpKy1trNEmJZDIjaiaIMLyRA=
        - --email-domain=*
        - --auth-logging=true
        - --request-logging=true
        - --standard-logging=true
        - --show-debug-on-error=true
        - --cookie-domain=.simpledemo.onap.org
        - --cookie-expire=12h
        - --cookie-samesite=lax
        - --whitelist-domain=.simpledemo.onap.org
        - --login-url=https://keycloak-ui.simpledemo.onap.org/auth/realms/ONAP/protocol/openid
        - --pass-access-token=true
        - --pass-authorization-header=true
        - --pass-host-header=true
        - --pass-user-headers=true
        - --http-address=0.0.0.0:4180
        - --oidc-email-claim=email
        - --oidc-groups-claim=groups
        - --insecure-oidc-skip-issuer-verification=true
        - --insecure-oidc-allow-unverified-email=true
        - --silence-ping-logging=true
        - --upstream=static://200
        - --set-xauthrequest=true
        - --set-authorization-header=true
      [...]
```

Most important settings are set-xauthrequest, set-authorization-header. This injects jwt tokens into headers that later can be used by istio or even upstream app. Now we can decide which token can be used

2.a Authorization Header

This allows to use groups out of the box (but nothing apart of groups can be used):

- create Istio resource RequestAuthentication:

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
```

```
kind: RequestAuthentication
metadata:
  name: echo-request-auth
  namespace: onap
spec:
  selector:
    matchLabels:
      app: echoserver
  jwtRules:
   - issuer: https://keycloak-ui.simpledemo.onap.org/auth/realms/ONAP
     jwksUri: http://keycloak-http.keycloak/auth/realms/ONAP/protocol/openid-connect/certs
```

What it does it tells Istio proxy on echoserver to look for jwt token in default location (Authorization Header) and extract from it all the claims. Additionally, if the token is invalid it will return 403 immediately to the user.

- create Istio Resource AuthorizationPolicy:

```
apiVersion: security.istio.io/v1
kind: AuthorizationPolicy
metadata:
  name: echo-authz-group-check-jwt
  namespace: onap
spec:
  action: ALLOW
  rules:
  - to:
    - operation:
        hosts:
        - echoserver.simpledemo.onap.org
        methods:
        - GET
    when:
    - key: request.auth.claims[groups]
      values:
      - onap_reader
  - to:
    - operation:
        hosts:
        - echoserver.simpledemo.onap.org
        methods:
        - GET
        - POST
        - PUT
    when:
    - key: request.auth.claims[groups]
      values:
      - admins
      - onap_admin
      - onap_contributor
  selector:
    matchLabels:
      app: echoserver
```

It extracts from claims the user groups and checks groups. This nicely allows us to limit access based on group, server, path and method

2.b User Info Header

This allows to use groups, roles, custom claims out of the box

- create Istio resource RequestAuthentication:

```
apiVersion: security.istio.io/v1beta1
kind: RequestAuthentication
metadata:
  name: echo-request-auth
  namespace: onap
spec:
  selector:
    matchLabels:
      app: echoserver
  jwtRules:
   - issuer: https://keycloak-ui.simpledemo.onap.org/auth/realms/ONAP
     jwksUri: http://keycloak-http.keycloak/auth/realms/ONAP/protocol/openid-connect/cert
     fromHeaders:
     - name: x-auth-request-access-token
```

The only difference here is that we specify what header host the jwt token, this token has many more claims so we can use much more later in Authentication Policy

- create Istio Resource AuthorizationPolicy:

```
apiVersion: security.istio.io/v1
kind: AuthorizationPolicy
metadata:
  name: echo-authz-group-check-jwt
  namespace: onap
spec:
  action: ALLOW
  rules:
  - to:
    - operation:
        hosts:
        - echoserver.simpledemo.onap.org
        methods:
        - GET
    when:
    - key: request.auth.claims[realm_access][roles]
      values:
      - onap_operator
  - to:
    - operation:
        hosts:
        - echoserver.simpledemo.onap.org
        methods:
        - GET
        - POST
        - PUT
    when:
    - key: request.auth.claims[realm_access][roles]
      values:
```

```
      - admin
      - onap_admin
      - onap_designer
  selector:
    matchLabels:
      app: echoserver
```

We can see that roles are used to grant or deny access to user, but any other claim (including custom ones) can be used.

Option 3: Rely on JWT Tokens Directly

Just rely on jwt tokens directly and forget about oauth2-proxy. It will be faster and more "genuine" but was not tested (it just came out to me only lately after reading: https://github.com/istio/istio/issues/8619)

Chapter 3: Open Questions

- JWT – token expires and is rejected by Istio with: "Jwt is expired." Fixes:
- Prolong JWT for longer than the cookie lasts (or reduce time for the cookie).
- Make re-login redirection in case of a 401 response, "Jwt is expired" from envoy. It can be done with EnvoyFilter.
- Make a custom theme for login.
- We should proceed more after an agreement with the customer:
  - We need to know IdP that will be used in the final solution; it may/will have different options than Keycloak.
  - Claim `aud` should be validated to forbid the abuse of the token.
  - We should have a bypass for OAuth2-Proxy, based on some header with a password, IP, TLS, etc. We must have the option to authorize traffic from tests or 3rd parties.