Pluggable Security

7. Pluggable User-level Authentication and Authorisation

Status: Draft

7.1 Background and Goals:

ONAP must be deployed in different service provider environments in order to be successful. We know that different service providers have requirements for different authentication and authorisation security infrastructures. Additionally while many use cases can be satisfied by component-to-component authentication and authorisation (e.g. orchestration activities) others need the identity and authority of the originating user (e.g. retrieval of data from a source with object level access control). To meet these requirements we need a security framework that is *pluggable*.

Since ONAP is a micro services architecture we must also address the standard requirements to localize the burdens of *configuration* and *patching.*

- **Goal 1**: Alternative Authentication and Authorisation security providers can be integrated without requiring customisation of the underlying ONAP code.
- **Goal 2**: Align with the existing AAF project, so as to promote re-use and to avoid duplication/fragmentation of the security architecture.
- **Goal 3**: Minimise the operational effort required to configure and patch microservices.
- **Goal 4**: Provide a solution that minimises the development impact on microservices written in Clojure, Python etc as well as Java
- **Goal 5:** Support human and non-person entities interacting with the ONAP applications at run-time.

Since CADI/AAF is our open-sourced Authorisation provider, we propose to build a reference implementation of pluggable authentication and authorisation based upon it.

Terminology:

Subject - An entity requesting to perform an operation upon the object. The subject is sometimes referred to as a requestor. The subject may be a human or a non-person entity.

User - Any subject that interacts with a system.

Non-person entity - A subject that is not a human.

Role - 1. A business responsibility that an employee or contractor fulfills.

2. A level of privilege assigned within a resource.

3. A defined set of user accounts for a job function.

7.2 Context

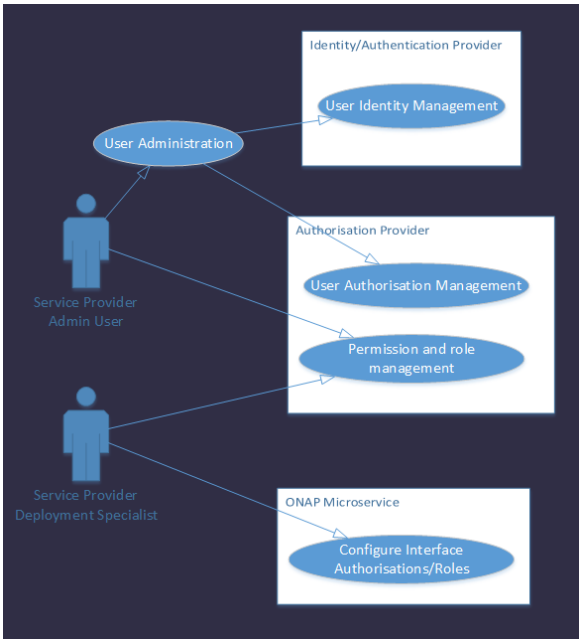
This proposal relies upon the ONAP Credential Management and ONAP Communication Security initiatives to provide:

1. Secure certificate generation and distribution.
2. Component->component trust through mutual end point authentication.
3. TLS communications resistant to:
    a. Spoofing
    b. Replay attacks
    c. Man-in-the-middle attacks
    d. Token theft.

This proposal relies upon the Service Provider's security infrastructure (Authorisation and Authentication) to provide:

1. User administration.
2. Permissions and role management support

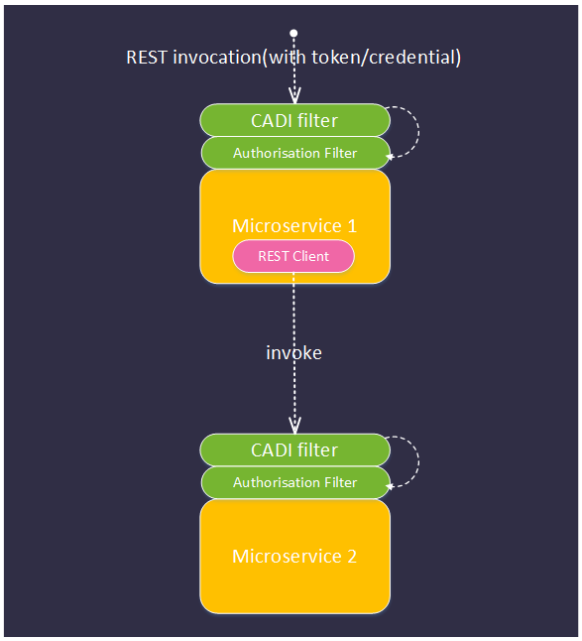
TO DO - insert general security arch diag



7.2.1 Proposal Summary

Overview

1. An upstream user or application invokes a REST endpoint on ONAP microservice (MS1) supplying one or more credentials or tokens (e.g. X509 cert, OAUTH2 token, SAML token, basic auth).
2. The request is intercepted by the CADI filter whose responsibilities include:
    a. extracting the required tokens/credentials from the request - configuration point 1
    b. invoking the correct authentication and authorisation providers to validate the supplied tokens and to discover their authorisations - configuration point 2 (See section 7.4.1 and 7.4.2)
    c. optionally caching the result of the authentication and authorisation steps above to optimise performance - configuration point 3 (see 7.4.4)
    d. rejecting requests with invalid credentials
    e. injecting the retrieved authorisation token(s) into a well-known place in the request (for further inspection by the authorisation filter).
3. The request is intercepted by the Authorisation Filter that:
    a. extracts the authorisation tokens from the request
    b. tests that the tokens have not been tampered with or expired
    c. compares the authorisations provided with those required to access the REST endpoint - configuration point 4.
    d. rejects requests that do not meet the authorisation criteria

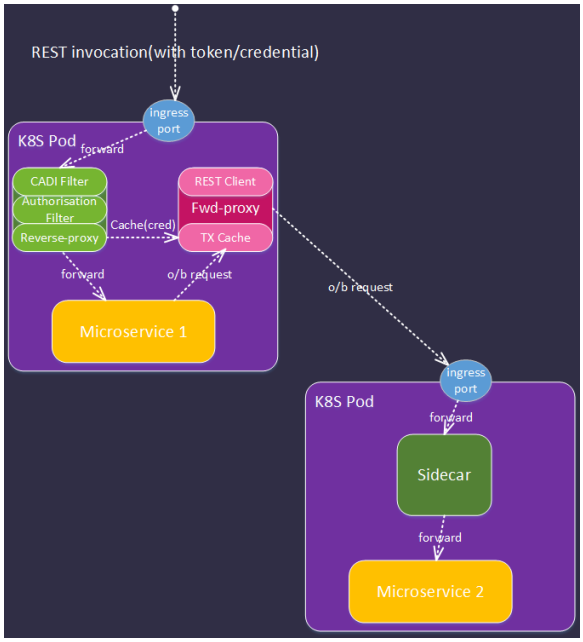


4. The request is passed to the application that performs business logic.
    a. The application logic may *optionally* programmatically query the authorisations associated with the request identity/identities (say to decide which menu options to present when serving a UI) See section 7.4.5.
    b. The application logic may *optionally* invoke REST requests on one or more microservices to perform its business. If it does so, it **must** use the provided REST client to ensure that the provided credentials/tokens are propagated correctly and securely. These tokens are not directly available to the application developer (see section 7.4.3)

Each subsequent microservice in the transaction performs the same sequence of operations.

7.2.2 SideCar implementation

In this implementation, we use the same security components outlined above, but externalised in a sidecar (K8S impl described):



1. An upstream user or application invokes a REST endpoint on ONAP microservice (MS1) supplying one or more credentials or tokens (e.g. X509 cert, OAUTH2 token, SAML token, basic auth).
2. The request arrives at the exposed ingress port of the K8s POD.
3. The port forwards the request to the reverse-proxy service.
4. The request is intercepted by the CADI filter as described in section 7.2.1
5. The request is intercepted by the Authorisation Filter as described in section 7.2.1
6. If the request is admitted the reverse proxy populates forward proxy tranasction (TX) cache with credentials (note if no TX on request, the reverse-proxy generates and appends to the request)
7. The admitted request is forwarded to primary service
8. If primary service makes o/b http requests these are forwarded to the forward proxy service via ip-tables.
    a. The forward proxy retrieves any credentials associated with the transaction and appends to the outgoing request
    b. If the TX is not in cache (e.g. request originates from primary MS on startup) only the primary client cert is available.
9. FUTURE - the forward proxy can offer a service (to the primary MS) that allows it to discover the authorisations associated with the transaction (similar to the 'is-user-in-role type function).

Each subsequent microservice in the transaction performs the same sequence of operations.

7.3 History

History of this proposal can be referenced here - Pluggable Security - history archive

7.4 Supported Interaction Patterns

7.4.1 CADI/AAF Authentication and Authorisation

Flow summary:

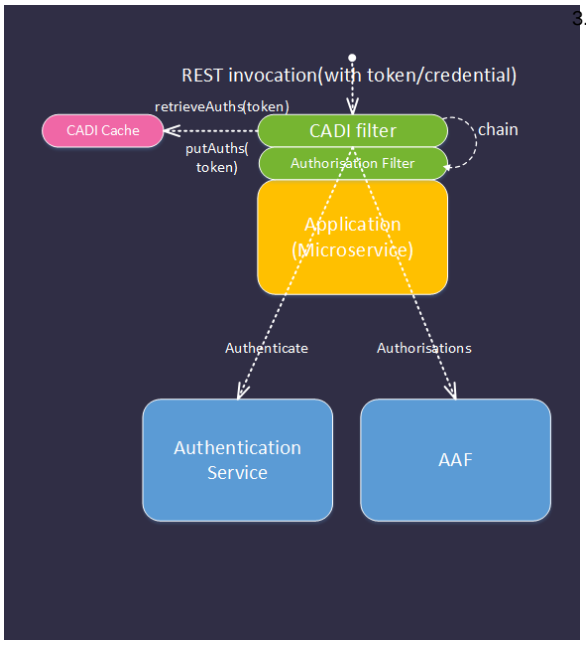
1.REST request arrives containing token(s) (or x509 cert subject)
2.CADI filter intercepts request and retrieves token:

1.If protocol/token type is configured for caching, CADI checks for match in cache

a. No match – validates token with supported authentication provider.
b. validation result is cached.
c. If token is not valid, request is rejected.

2.Retrieves authorisations from AAF

a. Caches result

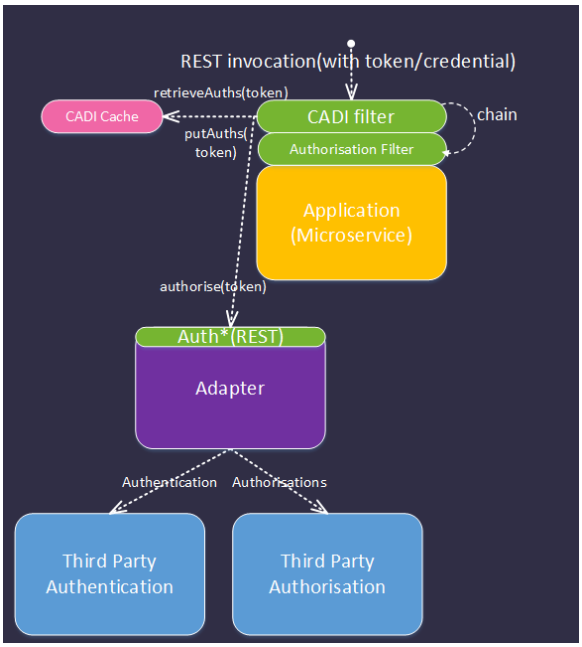


3. Authorisation filter performs admit/reject via authorisation policy:

1.The filter compares the authorisations retrieved by CADI with the configured requirements for the invoked method/URI pattern.

2. If the authorisations are satisfied, the filter admits the request. Otherwise the request is rejected with a 403.

7.4.2 CADI/3rd Party Authentication and Authorisation Provider



Flow summary:

1. REST request arrives containing token (or x509 cert subject)
2. CADI Filter is configured to extract tokens from one or more locations – e.g. headers, X509 cert subject

    1. If protocol/token type is configured for caching, CADI checks for match in cache

        a. No match – CADI is configured to forward tokens to external security microservice with AUTH* interface.
        b. Security Microservice implementation validates token with Authentication Provider
        c. Security Microservice implementation retrieves authorisations from Authorisation Provider
        d. Security microservice returns authorisations in standard format

    2. CADI Filter caches result

3. Authorisation filter performs admit/reject via authorisation policy:

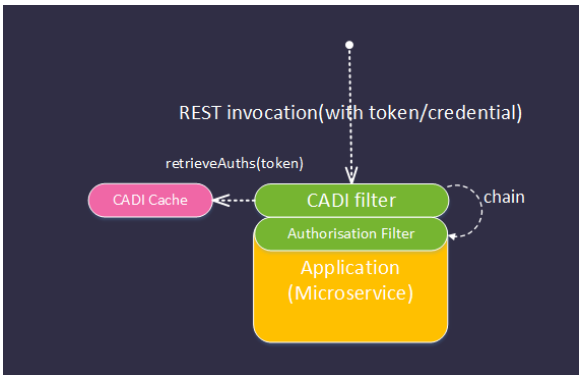
    1. The filter compares the authorisations retrieved by CADI with the configured requirements for the invoked method/URI pattern.

    2. If the authorisations are satisfied, the filter admits the request. Otherwise the request is rejected with a 403.

7.4.4 Authentication and Authorisation from Cached Result

Flow summary:

1. REST request arrives containing token (or x509 cert subject)
2. CADI Filter is configured to extract tokens from one or more locations – e.g. headers, X509 cert subject*
    a. Checks for match in cache
    b. Match found, identity is valid.
    c. Checks for matching authorisations in cache
    d. Match found
    e. Authorisations added to request
3. Authorisation filter performs admit/reject via authorisation policy:
    a. The filter compares the authorisations retrieved by CADI with the configured requirements for the invoked method/URI pattern.
    b. If the authorisations are satisfied, the filter admits the request. Otherwise the request is rejected with a 403.



7.4.3 Token Propagation between microservices

WIP

7.4.5 Programmatic Authorisation check

Done - to write up.

7.5 Configuration

The per-microservice configuration of pluggable security includes (but is not limited to):

1. JSON based CADI configuration, specifying which tokens are to be retrieved from an incoming REST request and the paths to them.
2. CADI configuration of the LUR and TAF interfaces to be used for authentication and authorisation and their respective REST endpoints - typically common for all microservices.
3. Configuration of the CADI caching mechanism (in memory, disk store, cache ageing policy) and configuration of which token types to cache.
4. JSON based Authorisation Filter configuration specifying the URI patterns for the microservice and the authorisations required to access them.

7.6 Identified Work Items

1. Describe Token propagation between filters
2. Authorisation Filter configuration - alignment with CADI configuration conventions
3. CADI configuration extensions:
    a. Describes what tokens/credentials to look for and where to find them in the REST request.
    b. Configure TAF/LUR to call Auth* endpoint.
4. Generic Security Provider TAF/LUR implementation to support **CADI/3rd Party Authentication and Authorisation Provider** use case.
5. Sample Auth* implementation for reference/OOB.
6. Enhancements to support serviceservice token propagation.
7. Updated client impact statement.
8. Updated service impact statement.

7.7 Impacts of Pluggable Security (WIP):

To ONAP clients (internal or external):

1. (development) Need to integrate client with Authentication Provider to retrieve credential. Note - this could be abstracted via a call to Auth* configured with a re-direct if required?
2. (development) Client needs to provide tokenised credential in REST request.

To ONAP Services:

1. (development) Need to include the CADI and Authorisation Filters in their filter chains.
2. (deployment) Need to configure CADI client.
3. (deployment) Need to configure authorisation requirements per URI, per service. Note that this can be as fine or coarse as the service provider's project requires.
4. (development) All new or modified microservices need to use the secured REST client to invoke other microservices for seamless and secure propagation of identity and authorisations.

7.7.1 Impacts of Pluggable Security (Side car approach)

Impact on primary service developers

1. Configure CADI and Authorisation filters (authorisation enforcement point)
2. Complete Helm chart for microservice POD
3. Regression test primary m/s.
4. Remove any pre-existing authorisation point in the primary microservice code.

7.8 Open Issues

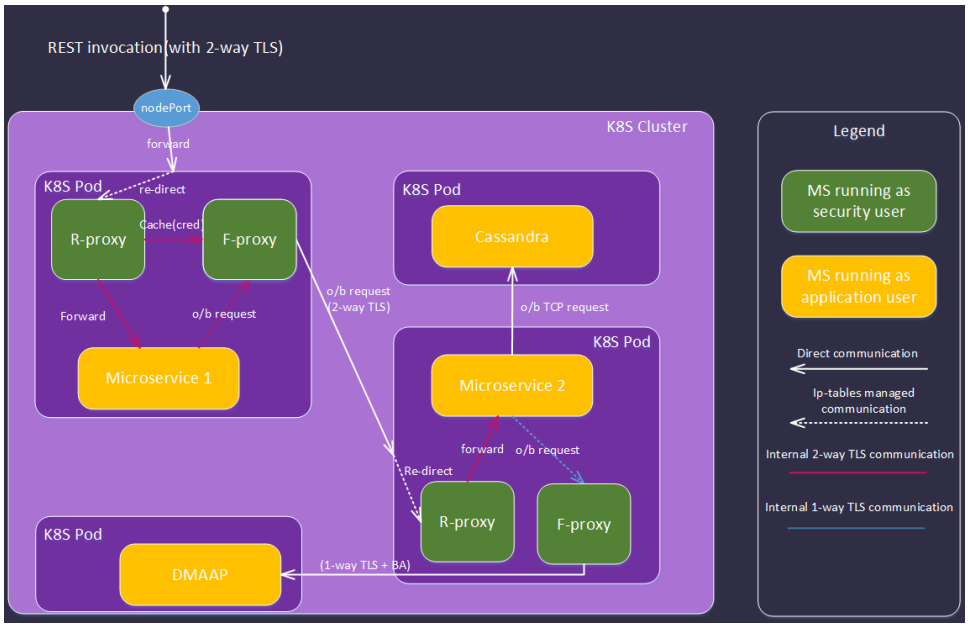
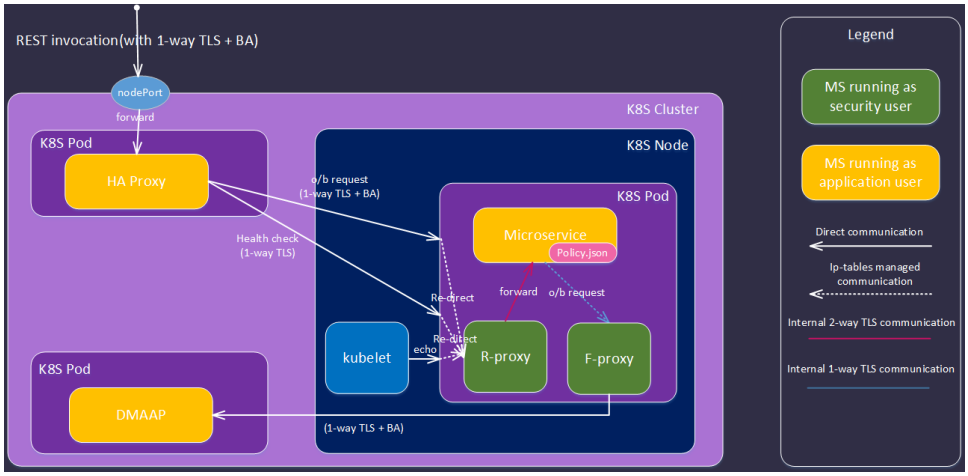
1. What happens in the case where there is an overlap between a protocol that CADI already supports and the generic Security Service config? How do we avoid conflicting/confusing configuration for the deployer (and the associated risk of leaving a gap/vulnerability)?
2. We need to agree on a common/preferred representation for the authorisation token(s).
3. Agree behaviour when no token is present/authentication fails - largely aligned.

7.9 Next Steps

1. Complete Use cases
2. Agree schedule/roadmap for work items (what can be done in Casablanca/after Casablanca)
3. Review/refine proposal with Seccom/TSC.

7.10 Identified and supported patterns and features

The table below describes patterns of behaviour that we have encountered whilst deploying the security sidecar with A&AI components. The table summarises the patterns and the status of support/testing.

<table>
<tr><th>Pattern /Feature</th><th>Description</th><th>Supported</th><th></th></tr>
<tr><td>DMAAP Integration</td><td>A Microservice needs to call out to DMAAP from within a sidecar-enabled POD</td><td>Yes - Dublin</td><td rowspan="3"> 
</td></tr>
<tr><td>Database Integration</td><td>A Microservice needs to call out to a Database from within a sidecar-enabled POD</td><td>Yes - Dublin</td></tr>
<tr><td>Application Credential propagation</td><td>Microservice 1 to Microservice 2 with 2 way TLS and application credential propagation. This is the archetypal use case for the side car.</td><td></td></tr>
<tr><td>Pod readiness probe</td><td>Kubernetes uses a readiness probe to check if an MS is up (say for dependency management). This can be implemented via an https GET request.</td><td>Yes - Dublin</td><td rowspan="3"> 
</td></tr>
<tr><td>Load balancer health check</td><td>Load balancer uses an HTTP poll to check if an MS is up</td><td>Yes - Dublin</td></tr>
<tr><td>Microservice has embedded authorisation policy file.</td><td>When pluggable security is deployed, the subject in the client cert sent by rproxy to resources is used to check against the</td><td>Yes - Dublin</td></tr>
</table>

	permissions configured in the amended policy json file.		
Spring security profile	3 profiles identified: **one-way-ssl** This uses one way ssl between the client and the target microservice and the authentication is Basic Auth. **aaf-auth** In this mode, the microservice communicates with AAF for auth via an active embedded CADI. **two-way-ssl** In this mode, the microservice uses the CN in client cert to match against a security policy file (e.g. aai_policy. json for authorization.	Yes - Dublin Untested Yes - Dublin	