Py-spiffe

Usage

1. As a py-spiffe library user, I want to fetch an X.509 SVID to identify myself (e.g. establish a TLS connection)
2. As a py-spiffe library user, I want to fetch X.509 bundles to validate the identity of other workloads, including workloads from federated trust domains.
3. As a py-spiffe library user, I want to fetch a JWT SVID to identify myself
4. As a py-spiffe library user, I want to fetch the JWT public key bundles to validate JWT tokens presented by other workloads.
5. As a py-spiffe user, I want to be able to establish mTLS connections using X509-SVIDs.
6. As a py-spiffe user on MacOs, I want to fetch the X.509 and JWT svids

Feature

with grpc.insecure_channel(NSPAddressLocal) as channel:Client API:

- JWT SVID functionality:
    - Fetches a SPIFFE JWT-SVID from Workload API on one-shot blocking call.
    - Fetches the JWT bundles for JWT-SVID validation, keyed by trust domain.
    - Validates the JWT-SVID token.
        - Parses and validates a JWT-SVID token using a set of JWT authorities (public keys) and a set of target audiences. Validate a JWT SVID for the following:
            - The peer presents a valid JWT-SVID
            - The peer presents a particular SPIFFE Id (‘aud’ claim)
    - Watches for JWT bundles updates.
        - Set a watcher on the Workload API to call a callback function when a jwt bundle rotation occurs
        - Retry when the connection fails
        - Additional error treatment

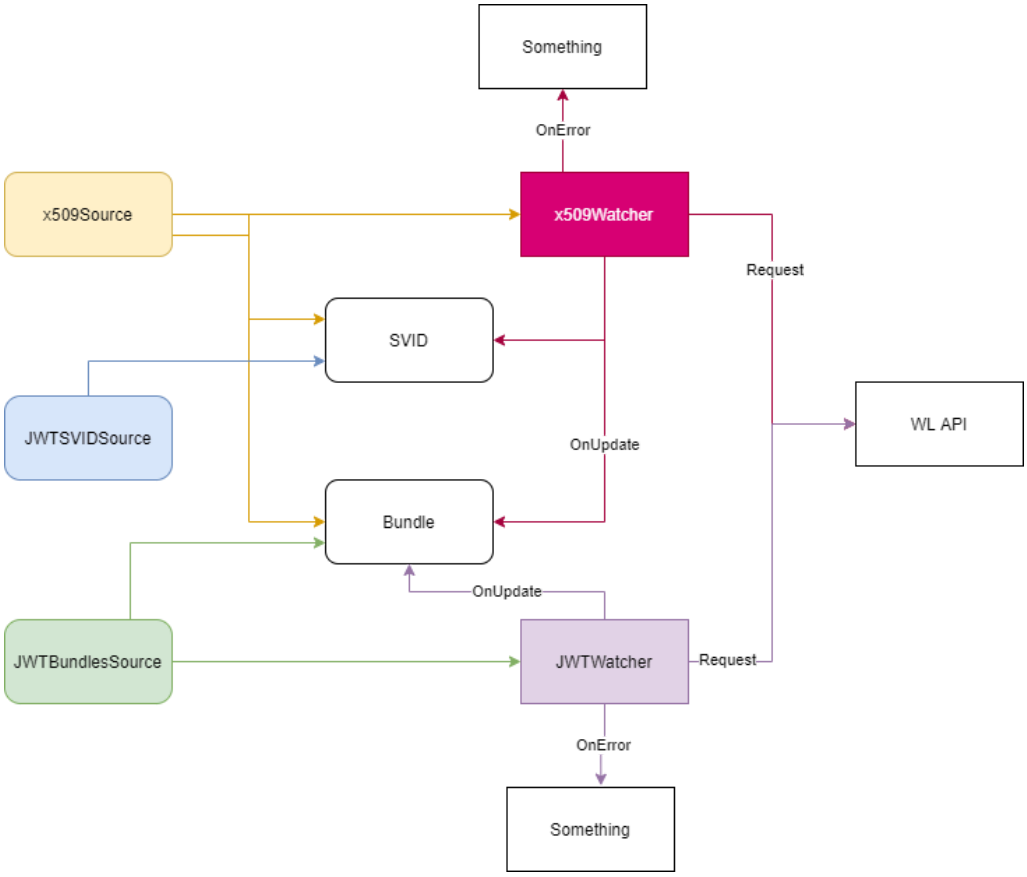
- X509 SVID functionality:
    - Fetches an X.509 context on a one-shot blocking call.
        - Parses an X.509 SVID (SpiffeId, certificate chain, private key) from a WorkloadAPI X509-SVID response

        - Parses or processes X.509 bundles from Workload API X509-SVID response
    - Validates a X.509 certificate chain using a set of X.509 Bundles
    - Watches for X.509 context updates.
        - Set a watcher on the Workload API that calls a callback function when an SVID rotation occurs.
        - Retry when the connection fails
        - Additional error treatment
- mTLS functionality
    - Connect to an mTLS peer using credentials fetched from the Workload API
        - **ISSUE 1**: loading certificate from memory. Only loading from disk is available in Python https://bugs.python.org/issue16487!!!
        - Possible solution: save the certificate and the key encrypted in disk, load it, delete it. The key to encrypt the key is generated in memory whenever needed.
        - **ISSUE 2**: (ssl.SSLContext.load_verify_locations) - will need to save in disk and this may be an issue
    - Create an mTLS listener using credentials fetched from the Workload API
    - Validate the peer for the following use cases:
        - The peer presents a valid SVID
        - The peer presents a SPIFFE Id in a particular trust domain.
        - The peer presents a particular SPIFFE Id
        - The peer presents a SPIFFE Id that matches an entry in a set of allowed SPIFFE Ids

Basic SPIFFE

- Validate a URI string as a correct SPIFFE Id
- Validate a URI socket endpoint address

High level abstractions



- Make use of PyJWT(preferred 1.7.1 but 2.0.0 is going to be released soon). It can be used for bundles and keys.
    - Version 1.5.1 has vulnerabilities reported so we should not allow the usage of this. Suggestion is to require 1.7.1 at a minimum.