Policy Enforcement Assessment

March 2024

1. Table of Contents:

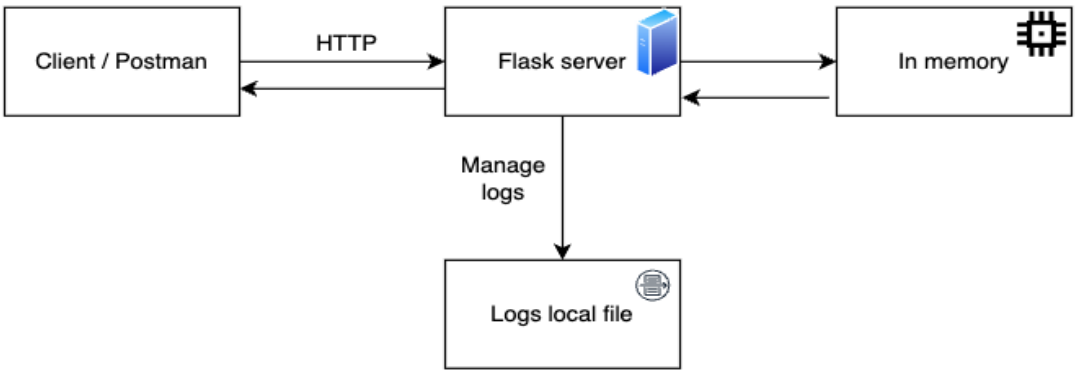
2. Introduction:

Policy enforcement is an important tool of network security is network policy management. This refers to defining what communication in a network is allowed or forbidden. These policies can monitor traffic and alert when something forbidden occurs or can be enforced using a firewall or NAC solution. The solution is a programmatic CRUD interface for managing network policies.

3. Project Overview:

A Python implementation of a programmatic CRUD interface for managing network policies and rules. The system allows defining policies and rules, enforcing restrictions, and performing various operations on them.

4. System Architecture:



5. Functional requirements:

Phase 1:

- Work on the code skeleton of stage 1.
- Initial API ability.
- Define Policy - With name and description.
- Data should be stored in memory and does not need to be saved to a persistent data store.
- All methods should take arguments and return values which are JSON strings.
- Managing errors.
- create_policy() should take as its only argument a JSON string containing an object whose keys are the fields of the policy, and should return a JSON string containing something that can be used to identify the policy in the future.
- list_policies() should return a JSON string containing an array of objects, each of which represents a policy and should have keys corresponding to each of the policy's fields.
- Pass the unit tests of stage 1.

Phase 2:

- Work on the code skeleton of stage 2.
- Update Policy - with type, "Arupa" or "Frisco".
- An Arupa policy may not have the same name as another Arupa policy. Multiple Frisco policies may have the same name. This requirement replaces the global policy name uniqueness requirement from the previous step.
- read_policy()- take as their first argument the same object identifiers that the create_policy() method returns. Return a JSON string containing an object in a format similar or identical to the objects in the response from the list_policies() method.
- update_policy() - take as their first argument the same object identifiers that the create_policy() method returns. take as its second argument the same input which is passed to the create_policy() method, supporting the same set of fields.
- delete_policy() - take as their first argument the same object identifiers that the create_policy() method returns.
- Pass the unit tests of stage 2.

Phase 3:

- Work on the code skeleton of stage 3.
- Define Rule - with name, ip_proto, source_port.
- Define the Arupa rule - Based on Rule with source_subnet.
- Define the Frisco rule - Based on Rule with source_ip, destination_ip.
- Arupa rule names must be unique within each policy.
- Frisco rule names must be globally unique, even between policies.
- get_rules() - take a policy identifier as its only argument, and should return all of the rules associated with that policy.
- get_rule() -

- create_rule() - take as its first argument a policy identifier to which the rule should belong, and as its second argument a JSON string containing an object whose keys are the fields of the rule.
- update_rule() -
- delete_rule() -

Phase 4: - Refactor and improvements.

- The rule can not be implemented if it is not Arupa or Frisco. Must be one of them.
- Names better management?
- Actions that can be reduced in complexity or amount of running.
- Tests for phase 3.

6. Non-Functional Requirements:

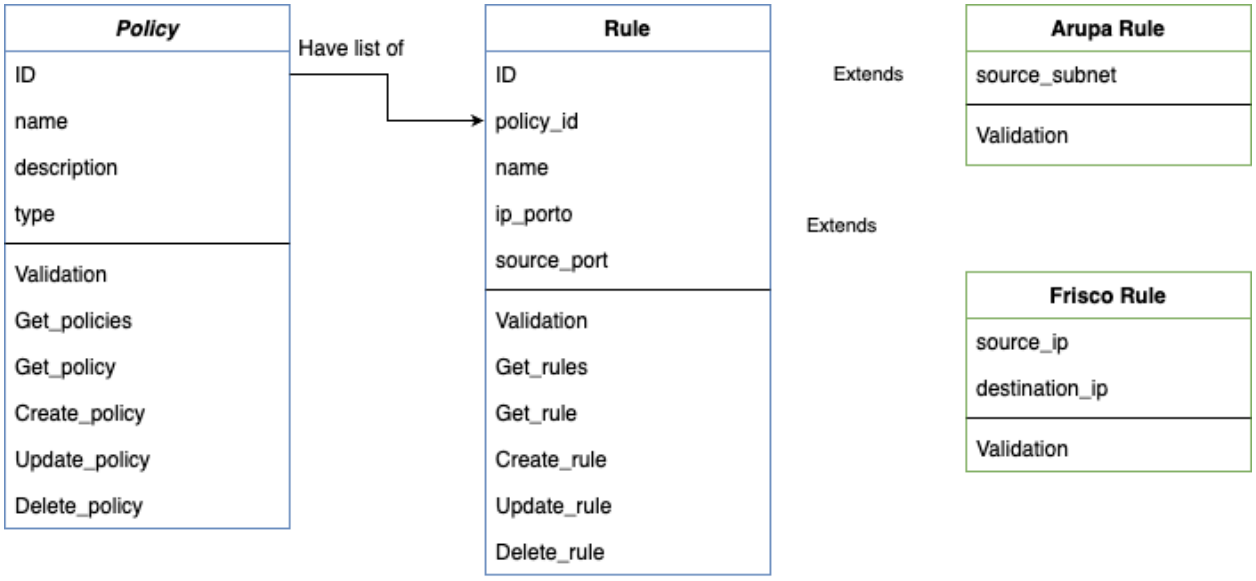
Packages that I use by requirements and by my opinion:

- Code language - Python ^3.6
- Style guideline - PEP8 - Style guide.
- Static types - mypy - Gradual static type checking.
- Data modeling - Pydantic - Data parsing and validation library
- Package managing - Poetry - Packaging and dependency manager
- collections - Standard library data structures
- Working with IP - ipaddress - Standard libary IP address manipulation
- Server for API -Flask.
- Testing - pytest - Testing framework.
- Clean code.

7. Technical Stack:

- Server - Flask.
- API - simple CRUD design.
- Code design - MVC.
- Memory - Inmemory of the local session.
- Error handling - high-level catch pattern and throw exceptions on validations.
- TDD.

8. Data Model:



- PolicyAPI: The main class responsible for managing policies and rules.
- Policy: This is a group of definitions of what traffic is allowed or forbidden.
- Rule: This is one of the definitions specifying a type of network traffic.

9. User Interface (UI) Design:

Endpoints:

- GET '/' - Application entry.
- GET '/policies' - Get policies.
- POST '/policies' - Create a new policy.
- GET '/policies/<policy_id>' - Get policy.
- PUT '/policies/<policy_id>' - Update policy.
- DELETE '/policies/<policy_id>' - Delete policy.
- GET '/policies/<policy_id>/rules' - Get policy rules.
- POST '/policies/<policy_id>/rules' - Create a new rule.
- GET '/policies/<policy_id>/rules<rule_id>' - Get policy rule.
- PUT '/policies/<policy_id>/rules<rule_id>' - Update policy rule.
- DELETE '/policies/<policy_id>/rules<rule_id>' - Delete policy rule.

10. Deployment Architecture:

None.

11. Testing Strategy:
Unit tests: Every stage has its unit tests to check the main behavior of the critical functions. The function has input and output that we can predict, there for the test are units and not components (no 3rd party that we need to mock).
Component tests: None.
Integration tests: None.
E2E tests: None.

12. Development Environment:
Local virtual environment.
- Python 3.6 or above
- Poetry
- Clone the repository to your local machine.
- Open a terminal window and navigate to the root directory of the cloned repository.

```
$ python3 -m venv venv
$ . venv/bin/activate
$ poetry install
```

13. Project Timeline:
1. Phase 1 + Infrastructure + project boundaries - 2d.
2. Phase 2 - 1d.
3. Phase 3 - 2d.
4. Phase 4 - 1d.

14. Risks and Mitigation:
- Deadlines.
- Missing requirements.

15. Maintenance and Support:
- Codebase - In Github - https://github.com/Daniel-Modilevsky/policy_enforcement/
- Phase 1 - https://github.com/Daniel-Modilevsky/policy_enforcement/pull/1/files
- Phase 2 - https://github.com/Daniel-Modilevsky/policy_enforcement/pull/2/files
- Phase 3 - https://github.com/Daniel-Modilevsky/policy_enforcement/pull/4/files

16. Documentation:
- Pedantic - https://docs.pydantic.dev/latest/
- Mypy - https://mypy.readthedocs.io/en/stable/
- IP address - https://docs.python.org/3/library/ipaddress.html#module-ipaddress