Policy Enforcement Assessment

Introduction

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 be used to monitor traffic and alert when something forbidden occurs, or can be enforced using a firewall or NAC solution.

Your task is to implement a programmatic CRUD interface for managing network policies. This task is inspired by the field of network security programming, but does not require specialized knowledge or experience of computer networking or security.

Stages of the assessment

The assessment has two stages, the second of which builds on the first. **It is OK to submit only the first stage.** More code doesn't mean a higher grade, it means you are giving us more input to base our assessment on. We don't want to lose you because you find the assessment too long, but we want to give you space to shine. When you reach a point that showcases your skills well, you may stop and submit. We encourage you to polish the first stage before moving on to the second.

Please submit the solution to each stage in a separate file (or set of files if you split up the implementation), even though the second stage may be based on code from the first (i.e, copy aside the solution to stage 1 before continuing to implement stage 2).

You have as much time as you want, but the entire assessment is estimated to take a few hours.

Instructions

The exercise stages build on each other, so all instructions and requirements given for the first stage apply to the second stage as well.

You are provided with a code skeleton on which to build, and a **limited** set of automated tests for stage 1 to make sure you're on the right track. When evaluating your submission we will run additional tests as well as review your code manually. We encourage you to write tests for your solution. You are welcome, but not required, to submit them.

Solutions should be implemented in Python. Any version of Python above 3.6 is acceptable. You are welcome to use any external packages available on pypi, but please clearly document the names and versions of all packages that your solution requires in order to run.

We're looking for solutions which fulfill the specification, but also which demonstrate an ability to write code which is clear, concise, robust, internally consistent, idiomatic, and well-designed. Your solution should reflect both your understanding of the problem and your skills as a developer. Treat the problem statement as a real-life scenario rather than an exercise.

The last section of this document contains a list of links to tools. None of them are mandatory, but they may help you if you need some direction. Feel free to use them in any way you find convenient, or not at all.

The exercise's complexity

This exercise is intended to recruit for senior positions, and we evaluate them as such. Unless you have good reason to believe that your Python level is high, (e.g. some years writing Python in a high-standards production environment) we *encourage you to avoid the impression that this exercise is trivial*.

If you think that you're done and that the exercise was easy, consider: - Do none of the code paths fail? - Is the specification fully reflected in the code? - Are any assumptions that deviate or improve on the spec documented with their justifications? - Were this a pull-request, would it have passed with little improvement comments? (and many impressed ones? :)) - Can the readability, performance, and reliability be improved? Is the style great? - Is the API that is being exposed well-designed? - How convinced am I that there's no better way to do this?

We want to see you do a great job, and would like our expectations to be mutually calibrated. Good luck!

Technical writing sample

In addition to this assessment, please send us a 1-2 paragraph technical description in English of a project you were involved in. We will talk to you about it after the assessment, and we want to prepare in advance for this talk.

Resources

- pytest - Testing framework
- PEP8 - Style guide
- PEP20 - The Zen of Python
- Flake8 - A linter for PEP8 and more
- mypy - Gradual static type checking
- Pydantic - Data parsing and validation library
- Poetry - Packaging and dependency manager
- collections - Standard library data structures
- itertools - Standard library iterators and looping
- more_itertools - A library that expands on `itertools`
- ipaddress - Standard libary IP address manipulation
- socket - Standard library low-level networking
- logging - Standard library logging
- OSI model - A conceptual model of network communication layering