Access-Control Automation Review

Example report

Kit Baroness | Security analysis example

Objective and source basis

My Python file-update project implements an algorithm for maintaining an IP-address allow list. The file identifies addresses approved for a restricted subnetwork; a separate removal list identifies addresses that should lose that approval. The work demonstrates reading the file, transforming its contents, removing matching entries, and writing the revised list.

The supplied project includes the complete algorithm and its explanation. This report reviews that implementation. It does not claim deployment to a live access-control service or successful access tests against a restricted network.

Implementation reviewed

The references identify sections of the supplied project.

Source reference	Implemented operation	Effect
PY-S1: file input	Open allow_list.txt in read mode within a with statement, then read its contents.	Obtain the current text; close the file when the block ends.
PY-S2: parsing	Split the text on whitespace.	Create individual address entries for comparison.
PY-S3: removal	Iterate over remove_list; remove an address when it is present.	Exclude the matching entries under the stated input assumptions.
PY-S4: file output	Join the remaining entries with newline separators, then open the file in write mode.	Replace the file contents with one retained address per line.

The supplied removal list contains three addresses. The algorithm checks membership before removal, so an address already absent does not cause that removal step to fail. Retained entries keep their relative order. The source explicitly assumes the allow list has no duplicate addresses: removing one matching item does not otherwise guarantee that every duplicate is removed.

Security and operational assessment

The transformation supports an access decision by maintaining its input data. Updating a text file alone does not establish that a consuming service has loaded the change or that effective access has been revoked.

The implementation rewrites the source file directly. The supplied record does not demonstrate malformed-input handling, concurrent-update protection, interrupted-write recovery, or deployment validation. Those are review and delivery considerations before using the algorithm in a consequential environment.

For a functional design, the proposed next step is to separate the address transformation from file reads and writes. That would allow the input/output rules to be checked independently of the effect that changes the file. This is an engineering improvement proposed by this review, rather than a claim about the original implementation.

Proposed validation and release conditions

Check	Expected result	Current record
Matching, absent, and retained addresses	Removed entries are absent; authorized entries remain in order.	Algorithm reviewed; executed test results not supplied.
Duplicate, empty, and malformed input	Behavior matches an explicit input policy; unexpected data produces a controlled result.	Duplicate-free assumption stated; wider handling needs definition.
File update and service consumption	Reviewed changes are applied completely, can be recovered, and are loaded by the intended service.	Operational validation not supplied.

Before release, the technical owner should retain a before-state, review a proposed diff, record actual tests, and confirm the downstream access result. Acceptance requires file integrity, service continuity, and an accountable decision on remaining limitations.

Reference method

Change control and impact analysis are informed by NIST SP 800-128 and SP 800-53 Rev. 5 CM-3 and CM-4. Access enforcement and validation are supported by the selected AC-3 and SP 800-53A concepts.

https://csrc.nist.gov/pubs/sp/800/128/upd1/final

https://doi.org/10.6028/NIST.SP.800-53Ar5