REPORT
Python Algorithm for Managing IP Access

v1.0.1

Author:

Eldon Gabriel

July 11, 2025

TABLE OF CONTENTS

REVISION HISTORY

Version	Date	Author	Description of Changes
v1.0.0	01/01/2025	Eldon G.	Initial draft.
v1.0.1	07/11/2025	Eldon G.	Structured full report with section headings and conclusion.

SECTION 1.0: PYTHON ALGORITHM FOR MANAGING IP ACCESS

1.1 Project Description

In this project, I developed a Python algorithm to manage access control by updating an allow list of IP addresses. The algorithm reads a file containing allowed IPs, removes any that appear on a separate remove list, and updates the file accordingly. This ensures that employees no longer authorized to access restricted data are promptly removed from the allow list.

1.2 Open the File That Contains the Allow List

To open the `allow_list.txt` file, I used the `with open()` statement, which ensures proper file handling. Assigning the filename to the `import_file` variable makes it easier to reference later.

```
import_file = "allow_list.txt"
with open(import_file, "r") as file:
    ip_addresses = file.read()
```

- `with open(import_file, "r") as file:` opens the file in read mode.
- `file.read()` reads the contents of the file into a string.
- Using `with` ensures the file is properly closed after reading.

1.3 Read The File Contents

Once the file is opened, the `.read()` method is used to store its contents in the `ip_addresses` variable as a string. This allows for further processing.

```
ip_addresses = file.read()
```

- `.read()` reads the entire file into a single string.
- Storing it in `ip_addresses` makes it easy to manipulate later.

1.4 Convert The String Into A List

Since the IPs need to be checked and removed individually, they must be stored in a list format. The `.split()` method is used to achieve this:

```
ip_addresses = ip_addresses.split("\n")
```

- `.split("\n")` converts the string into a list, using newline characters as delimiters.
- Each IP address now becomes an element in the list.

1.5 Iterate Through The Remove List

A separate list called `remove_list` contains the IPs to be removed. A `for` loop is used to iterate through this list:

```
remove_list = ["192.168.1.10", "10.0.0.5"]
for element in remove_list:
    if element in ip_addresses:
        ip_addresses.remove(element)
```

- `for element in remove_list:` iterates through each IP in `remove_list`.
- `if element in ip_addresses:` checks if the IP exists in `ip_addresses`.
- `.remove(element)` removes it if found.
- This method works effectively because `ip_addresses` does not contain duplicates.

1.6 Remove IP Addresses That Are On The Remove List

This step ensures that all IP addresses found in `remove_list` are deleted from `ip_addresses` to maintain access control.

```
for element in remove_list:
    if element in ip_addresses:
        ip_addresses.remove(element)
```

- Ensures only authorized users remain in the allow list.
- Prevents unauthorized access by removing unwanted IPs.

1.7 Update File With The Revised List Of IP Addresses

After modifications, the updated IP list must be written back to the file. The `.join()` method is used to convert the list back into a string before writing:

```
with open(import_file, "w") as file:
    file.write("\n".join(ip_addresses))
```

- `.join(ip_addresses)` converts the list back into a newline-separated string.
- Opening the file in write mode (`"w"`) overwrites it with the updated contents.

SECTION 2.0: CONCLUSION

2.1 Key Takeaways

- Python can automate access control by managing allow lists programmatically.
- Reading, parsing, and rewriting IP files ensures policy enforcement at scale.
- Using lists allows efficient iteration and conditional removal of IPs.
- Regular updates to allow lists reduce the risk of unauthorized access.
- Secure file handling `(with open)` prevents data loss and improves reliability.