Algorithm for File Updates in Python

Project Description

In this project, I developed a Python algorithm to manage and update an allow list of IP addresses that control access to restricted content within an organization. As a security analyst, maintaining accurate access controls is critical to protecting sensitive information.

The algorithm reads a text file containing approved IP addresses, compares them against a list of IP addresses that should no longer have access, removes any unauthorized addresses, and updates the original file. This process automates access control maintenance and reduces the risk of human error.

Open the File That Contains the Allow List

To begin the algorithm, I assigned the file name "allow_list.txt" to a variable and used a with statement combined with the open() function. The with statement ensures the file is properly opened and automatically closed after the operation is complete. The file is opened in read mode ("r") so its contents can be accessed safely.

```
import_file = "allow_list.txt"

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

Read the File Contents

The .read() method converts the contents of the allow list file into a single string. This allows the data to be processed programmatically rather than manually reviewing the file. The contents are stored in the variable ip_addresses for further manipulation.

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

Convert the String into a List

To work with individual IP addresses, the string must be converted into a list. I used the .split() method, which separates the string based on whitespace and creates a list where each element is a single IP address. This makes iteration and comparison possible.

```
ip_addresses = ip_addresses.split()
```

Iterate Through the Remove List

A second list, remove_list, contains IP addresses that should no longer have access. I used a for loop to iterate through the list of IP addresses and evaluate each entry individually.

```
for element in ip_addresses:
   print(element)
```

Remove IP Addresses That Are on the Remove List

Inside the loop, I implemented a conditional statement to check whether the current IP address exists in the remove_list. If it does, the .remove() method deletes it from the allow list. This approach works because the list does not contain duplicate IP addresses.

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

Update the File with the Revised List of IP Addresses

After removing unauthorized IP addresses, the list must be converted back into a string so it can be written to the file. I used the .join() method to combine the list into a newline-separated string.

The file is then reopened in write mode ("w"), which overwrites the original contents with the updated allow list.

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

with open(import_file, "w") as file:
```

file.write(ip_addresses)

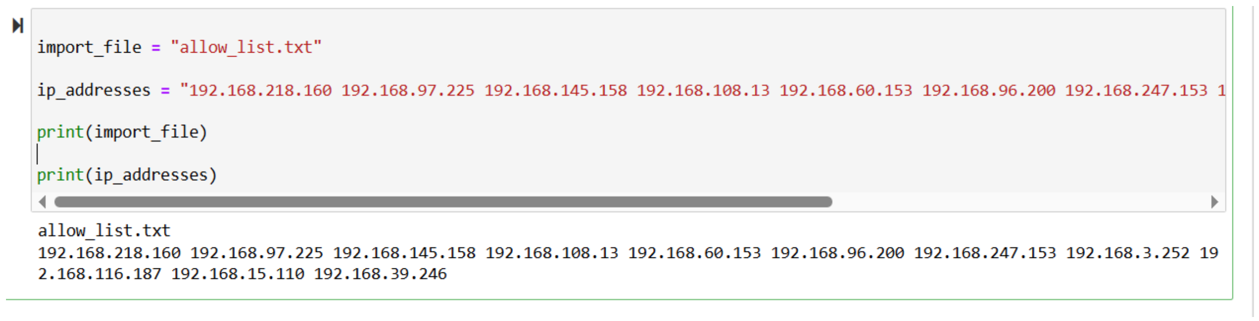
Summary

This algorithm demonstrates how Python can be used to automate access control management in a security environment. It uses file handling, string manipulation, conditional logic, and iteration to efficiently maintain an allow list of IP addresses. By automating this process, security teams can ensure that only authorized systems retain access to restricted resources.

Supporting Code Screenshots

Reading the Allow List File Securely

This code uses a `with` statement and the `open()` function to safely open the allow list file in read mode. Using a context manager ensures the file is properly closed after access, which is a best practice when handling security logs.

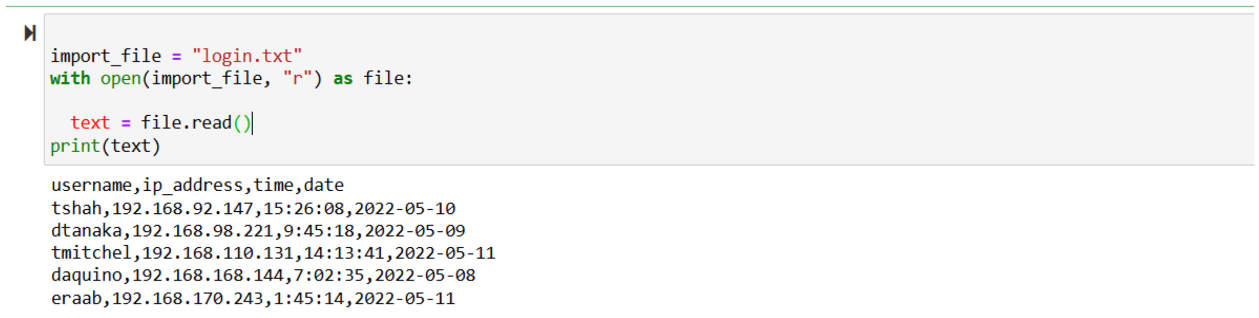


```
import_file = "allow_list.txt"

ip_addresses = "192.168.218.160 192.168.97.225 192.168.145.158 192.168.108.13 192.168.60.153 192.168.96.200 192.168.247.153 1

print(import_file)

print(ip_addresses)
```

```
allow_list.txt
192.168.218.160 192.168.97.225 192.168.145.158 192.168.108.13 192.168.60.153 192.168.96.200 192.168.247.153 192.168.3.252 19
2.168.116.187 192.168.15.110 192.168.39.246
```

Parsing File Contents into a String

The `.read()` method is used to load the contents of the allow list file into a string. This allows the log data to be processed and analyzed programmatically instead of being handled manually.

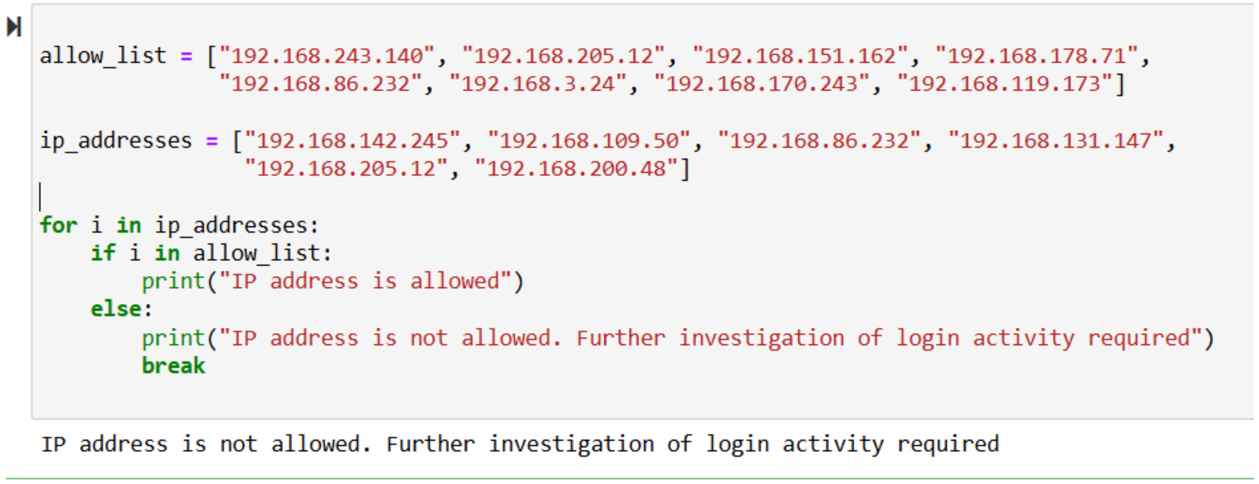


```
import_file = "login.txt"
with open(import_file, "r") as file:

  text = file.read()
print(text)
```

```
username,ip_address,time,date
tshah,192.168.92.147,15:26:08,2022-05-10
dtanaka,192.168.98.221,9:45:18,2022-05-09
tmitchel,192.168.110.131,14:13:41,2022-05-11
daquino,192.168.168.144,7:02:35,2022-05-08
eraab,192.168.170.243,1:45:14,2022-05-11
```

Extracting IP Addresses Using Regular Expressions

A regular expression is used to extract valid IP address patterns from log data. This approach mirrors security workflows such as identifying indicators of compromise (IOCs) in logs.

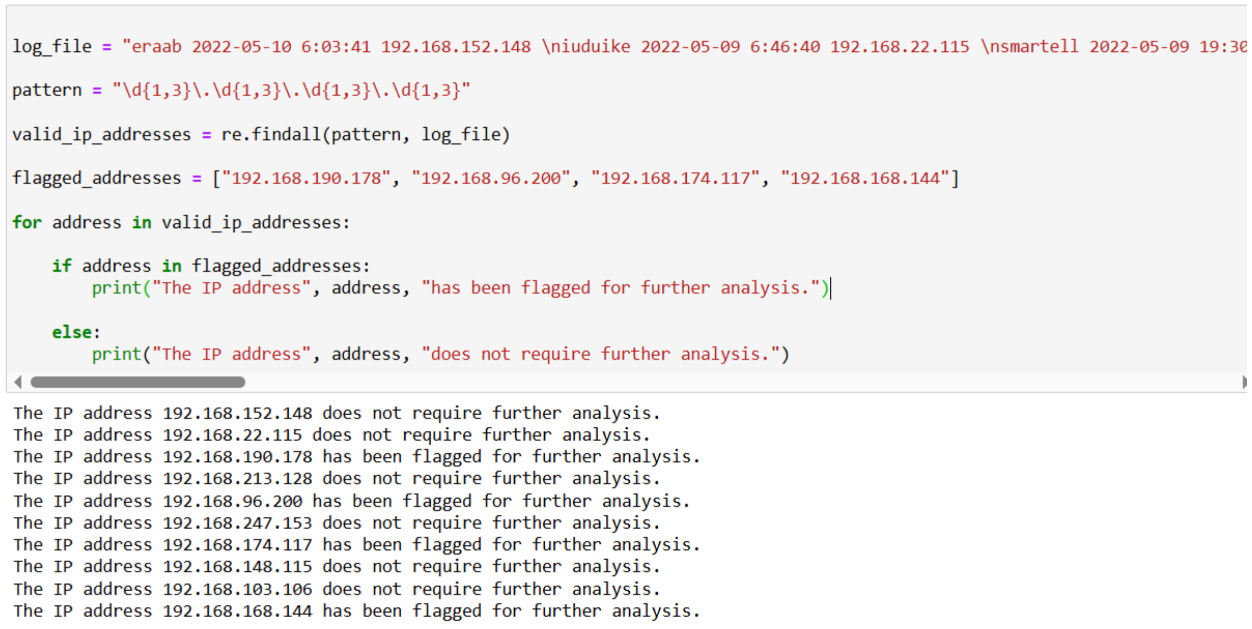


```python
allow_list = ["192.168.243.140", "192.168.205.12", "192.168.151.162", "192.168.178.71",
              "192.168.86.232", "192.168.3.24", "192.168.170.243", "192.168.119.173"]

ip_addresses = ["192.168.142.245", "192.168.109.50", "192.168.86.232", "192.168.131.147",
                "192.168.205.12", "192.168.200.48"]

for i in ip_addresses:
    if i in allow_list:
        print("IP address is allowed")
    else:
        print("IP address is not allowed. Further investigation of login activity required")
        break
```

```
IP address is not allowed. Further investigation of login activity required
```

Comparing Allow List and Remove List IP

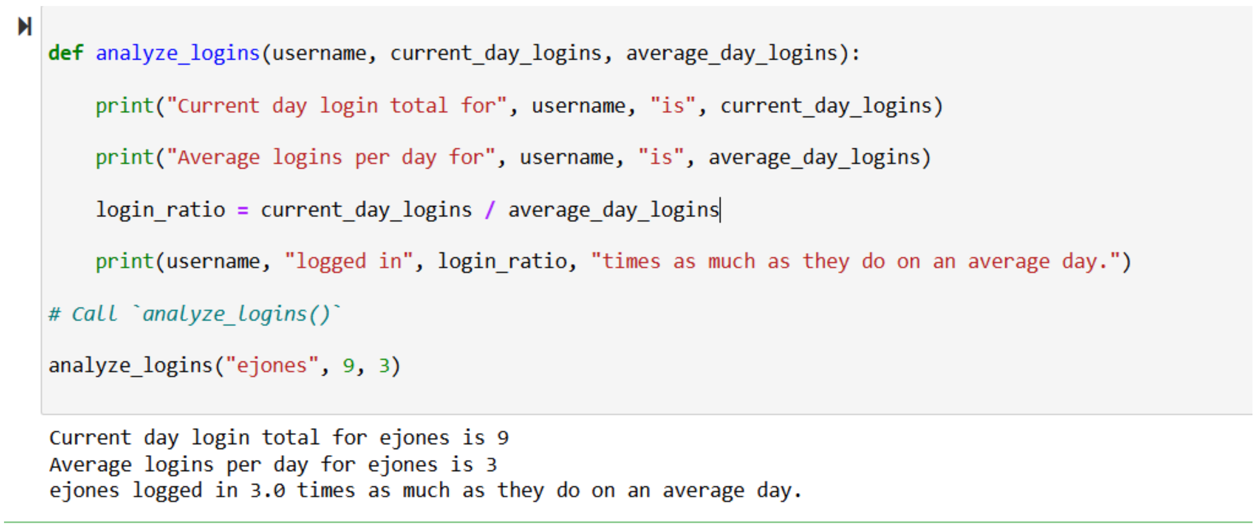
A loop and conditional logic are used to compare IP addresses in the allow list against a remove list. This enables automated identification of IP addresses that should no longer have access to restricted resources.



```python
log_file = "eraab 2022-05-10 6:03:41 192.168.152.148 \niuduike 2022-05-09 6:46:40 192.168.22.115 \nsmartell 2022-05-09 19:30

pattern = "\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}"

valid_ip_addresses = re.findall(pattern, log_file)

flagged_addresses = ["192.168.190.178", "192.168.96.200", "192.168.174.117", "192.168.168.144"]

for address in valid_ip_addresses:

    if address in flagged_addresses:
        print("The IP address", address, "has been flagged for further analysis.")

    else:
        print("The IP address", address, "does not require further analysis.")
```

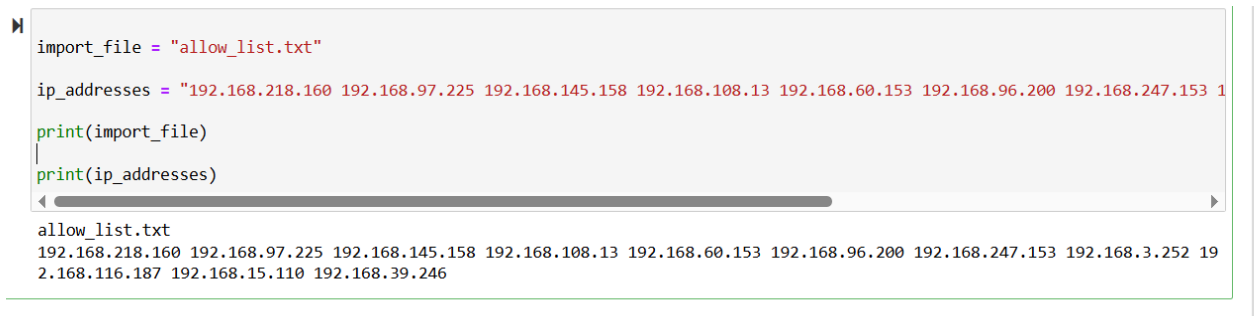
```
The IP address 192.168.152.148 does not require further analysis.
The IP address 192.168.22.115 does not require further analysis.
The IP address 192.168.190.178 has been flagged for further analysis.
The IP address 192.168.213.128 does not require further analysis.
The IP address 192.168.96.200 has been flagged for further analysis.
The IP address 192.168.247.153 does not require further analysis.
The IP address 192.168.174.117 has been flagged for further analysis.
The IP address 192.168.148.115 does not require further analysis.
The IP address 192.168.103.106 does not require further analysis.
The IP address 192.168.168.144 has been flagged for further analysis.
```

Automating File Updates with a Python Function This function encapsulates the file parsing, comparison, and update logic into a reusable Python function. Using a function improves code readability, maintainability, and makes the automation easy to reuse across systems.



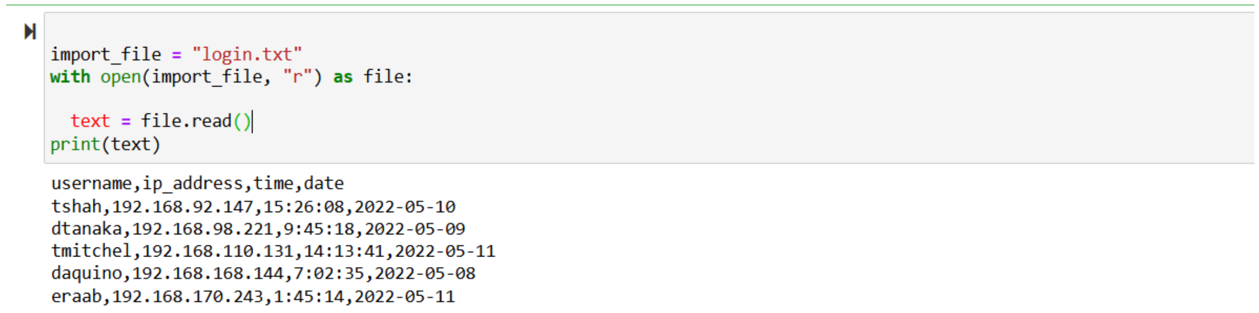
```
def analyze_logins(username, current_day_logins, average_day_logins):

    print("Current day login total for", username, "is", current_day_logins)

    print("Average logins per day for", username, "is", average_day_logins)

    login_ratio = current_day_logins / average_day_logins

    print(username, "logged in", login_ratio, "times as much as they do on an average day.")

# Call `analyze_logins()`

analyze_logins("ejones", 9, 3)
```

```
Current day login total for ejones is 9
Average logins per day for ejones is 3
ejones logged in 3.0 times as much as they do on an average day.
```



```
import_file = "allow_list.txt"

ip_addresses = "192.168.218.160 192.168.97.225 192.168.145.158 192.168.108.13 192.168.60.153 192.168.96.200 192.168.247.153 1

print(import_file)

print(ip_addresses)
```

```
allow_list.txt
192.168.218.160 192.168.97.225 192.168.145.158 192.168.108.13 192.168.60.153 192.168.96.200 192.168.247.153 192.168.3.252 19
2.168.116.187 192.168.15.110 192.168.39.246
```

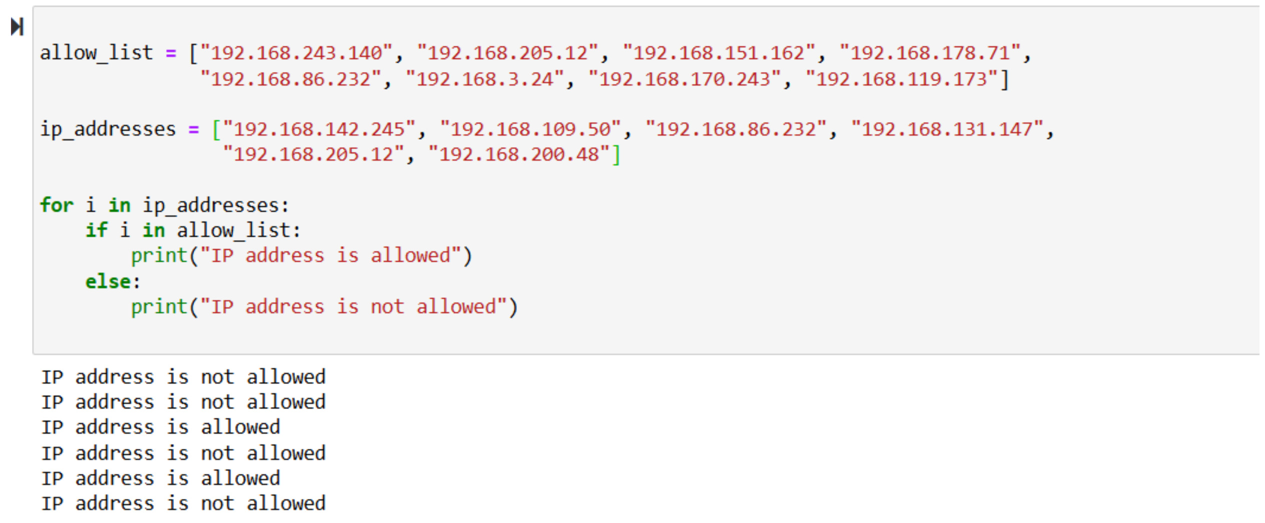


```
import_file = "login.txt"
with open(import_file, "r") as file:

  text = file.read()
print(text)
```

```
username,ip_address,time,date
tshah,192.168.92.147,15:26:08,2022-05-10
dtanaka,192.168.98.221,9:45:18,2022-05-09
tmitchel,192.168.110.131,14:13:41,2022-05-11
daquino,192.168.168.144,7:02:35,2022-05-08
eraab,192.168.170.243,1:45:14,2022-05-11
```

Managing Control Flow During Iteration

This code demonstrates how control flow can be managed during iteration using conditional logic and early exit behavior. By applying loop conditions effectively, the algorithm avoids unnecessary processing and improves efficiency when handling lists of IP addresses in a security context. This pattern is useful when responding to access-control decisions or stopping analysis once a condition is met.



```
allow_list = ["192.168.243.140", "192.168.205.12", "192.168.151.162", "192.168.178.71",
              "192.168.86.232", "192.168.3.24", "192.168.170.243", "192.168.119.173"]

ip_addresses = ["192.168.142.245", "192.168.109.50", "192.168.86.232", "192.168.131.147",
                "192.168.205.12", "192.168.200.48"]

for i in ip_addresses:
    if i in allow_list:
        print("IP address is allowed")
    else:
        print("IP address is not allowed")
```

```
IP address is not allowed
IP address is not allowed
IP address is allowed
IP address is not allowed
IP address is allowed
IP address is not allowed
```

Prepared by: Sambou Kamissoko
LinkedIn: https://www.linkedin.com/in/sambouk/