Algorithm for file updates in Python

Project description

In my role as a security professional within a healthcare company, my responsibilities encompass the regular maintenance and updating of files that dictate the access privileges of employees to restricted content. The core objective of this task revolves around identifying personnel authorized to access confidential patient records. This authorization is contingent upon their IP addresses. Our primary duty is to ensure the accuracy and relevance of the access list by meticulously removing the IP addresses of employees who are no longer affiliated with the company or those who have transitioned out of the relevant department. By executing these vital updates, we fortify the security framework and uphold the integrity of patient data, thereby fostering a secure and compliant operational environment.

Open the file that contains the allow list

```
# Assign 'import file' to the name file that contains the allow users:
import_file = "allow_list.txt"

# The 'with statement'
# Use 'open()' to import the allow user file and store it as a string
with open(import_file, "r") as file:
```

In the process of accessing and managing the **“allow_list.txt”** text file, the security analyst follows a systematic series of steps. Initially, the **import_file** variable is introduced, slated to house the data retrieved from the **“allow_list.txt”** file. The ensuing step employs the **with** keyword, enabling streamlined resource management during file interactions. Within this context, the **open()** function assumes a pivotal role, providing Python the means to initiate file access. This function mandates two parameters to fulfill its functionality. The initial parameter, **import_file**, serves as a reference to the target file for opening. The secondary parameter, designated as **“r”** to signify reading, specifies the intended operation with the file. Employing the “**as**” keyword, a variable is designated to symbolize a reference to another object, in this instance, labeled as **“file”**. This systematic approach culminates in the establishment of a structured environment for opening and engaging with the **“allow_list.txt”** file, allowing for efficient data retrieval and manipulation.

Read the file contents

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

    # Use '.read()' to read imported file and store in a variable named 'ip_addresses
    ip_addresses = file.read()
```

The security analyst addresses the process of preparing a file for reading, an essential prerequisite for subsequent operations. The procedure is initiated by implementing a **`with`** statement, a construct that encapsulates the operations to be performed within a controlled context. The **`open()`** function is integral to this step, with predefined parameters such as **`import_file`** representing the file under consideration, and the second parameter **`"r"`** signifying the intention to read the file's contents. Employing the "**`as`**" keyword establishes a variable as a reference to the file object, which streamlines subsequent interactions. Subsequent to the comment section outlining forthcoming actions, a new variable termed **`"ip_addresses"`** is introduced to house the content imported from the designated file. The "**`file.read()`**" function is then invoked, facilitating the opening and execution of the specified action. Concluding this process, the **`with`** statement ensures the file is closed, contributing to code readability and maintaining efficient file handling practices. This meticulous approach enhances the ability to effectively process and analyze the file's contents.

Convert the string into a list

```
# 'Use '.split()' to convert ip_addresses file from string to list.
ip_addresses = ip_addresses.split()
```

In this phase, the security analyst undertakes the task of converting a string into a more readable list format, a procedure commonly referred to as parsing. The pivotal tool employed for this transformation is the **`.split()`** function. By utilizing this function, the contents of the designated file variable, which in this instance is labeled as **`"ip_addresses"`**, are subjected to parsing. Upon execution, this process effectively transforms the **`"ip_addresses"`** file from its initial string state into a more structured and manageable list format, thereby enhancing its readability and usability for subsequent operations. This conversion contributes to the overall effectiveness of the code, enabling smoother processing and manipulation of the IP address data.

Iterate through the remove list

```
# Use 'for' loop to iterate through 'remove_list
# Use element as the loop variable
for element in remove_list:
```

In this particular phase, the security analyst embarks on iterating through the supplementary list named **`"remove_list"`**, encompassing a compilation of IP addresses earmarked for removal

from the **“allow_list.txt”** file. For a deeper comprehension, the **for** loop emerges as the pivotal construct of choice. This loop construct facilitates the automated and repetitive execution of code contingent on a predetermined sequence. The variable designated for the loop is **element**, while the source of iteration corresponds to the content within the **“remove_list”** file. By employing this configuration, the **element** iterates systematically through the **“remove_list”**, progressively appending the designated list of IP addresses slated for removal into the specified variable. This systematic iteration process ensures the precise identification and handling of IP addresses that necessitate removal, thereby refining the access control mechanism.

Remove IP addresses that are on the remove list

```
# Use conditional statement if 'element' is in 'ip_addresses
if element in ip_addresses:

    #Use '.remove()' method to remove 'elements' from 'ip_addresses'
    ip_addresses.remove(element)
```

In this section of the code, the security analyst focuses on the removal of specific IP addresses from the **ip_addresses** list, utilizing the variable **element** for this purpose. The process begins with the **“if”** keyword, initiating a conditional statement that assesses whether the provided code adheres to predetermined conditions. In this context, **element** represents the variable under consideration, while **“ip_addresses”** holds the data being evaluated. The colon “**:**” denotes the conclusion of the header, and its presence is crucial to prevent syntax errors. Subsequent indented lines are indicative of code that belongs to the body of the statement. The **.remove()** function is applied here, tasked with eliminating occurrences of the specified element from a list. In this specific instance, **element** encapsulates the content of the remove_list file, resulting in the removal of IP addresses from the **ip_addresses** list that are present in **“allow_list.txt”** but should not be retained. This systematic approach ensures the accurate management of access privileges and data integrity.

Update the file with the revised list of IP addresses

```
# Convert 'ip_addresses' back to string to be written into text file.
ip_addresses = "\n".join(ip_addresses)

# Create 'with' statement to write over 'import_file' variable.
with open(import_file, "w") as file:

    #Rewrite file to replace its content to 'ip_addresses'.
    file.write(ip_addresses)
```

In this process, the security analyst is tasked with updating a file containing a revised list of IP addresses. To achieve this, the analyst initially converts the **ip_addresses** variable from a list

format to a string, allowing for writing to a text file. The **“\n”** escape sequence is employed to establish new lines within the file. The **.join()** function is then utilized, facilitating the conversion of the list into a string format, with the “ip_addresses” content specified as the parameter. Subsequently, a **with** statement is employed in conjunction with the “open()” function, which takes two parameters: the target file (**“import_file”**) and the **“w”** mode for overwriting file content. The resulting new file is assigned using the “**as**” keyword. Finally, the file is rewritten using the **.write()** function, effectively replacing the existing IP addresses to eliminate unauthorized access to restricted content. This systematic approach ensures the accurate and secure updating of access privileges.

Summary

The security analyst team conducted a comprehensive examination of the**”allow_list.txt”** file, identifying and eliminating employee IP addresses that were no longer relevant due to departures or departmental transfers. A comparison was conducted with a separate **“remove_list”** file, aiding in the identification of employees no longer requiring access to restricted content outlined in **“allow_access.txt.”** Once the list was validated, necessary adjustments were implemented to ensure the updated user roster accurately reflected those with proper access privileges. This proactive approach safeguards sensitive information and maintains effective access control measures.