Algorithm for file updates in Python

-MY RESPONSES

Project description

I believe that automating the updating of access control files can improve security and efficiency. At my organization, access to restricted content is controlled with an allow list of IP addresses stored in the file "allow_list.txt". I believe that using a Python algorithm to remove IP addresses on a separate remove list ensures that only approved IPs maintain access.

More detailed of the Scenario:

You are a security professional working at a health care company. As part of your job, you're required to regularly update a file that identifies the employees who can access restricted content. The contents of the file are based on who is working with personal patient records. Employees are restricted access based on their IP address. There is an allow list for IP addresses permitted to sign into the restricted subnetwork. There's also a remove list that identifies which employees you must remove from this allow list.

Your task is to create an algorithm that uses Python code to check whether the allow list contains any IP addresses identified on the remove list. If so, you should remove those IP addresses from the file containing the allow list.

Note: This scenario involves developing the same algorithm that is developed in Tasks 2-7 of the Create another algorithm
lab. (You do not need to reference Task 1 and Tasks 8-10 of the lab to complete this portfolio activity.) You should revisit the lab to get screenshots to include in your portfolio document.

Open the file that contains the allow list

I believe that the first step in my algorithm is opening the "allow_list.txt" file. I assigned the file name as a string to the variable import_file. I believe that using a with statement along with the .open() function in read mode allows me to efficiently access the file while automatically managing its resources. I believe that the open() function takes two parameters: the first identifies the file to import, and the second specifies the mode. In this case, "r" indicates that I want to read the file. I believe that using as file assigns the file object to the variable file, which I can then use to interact with the file within the with statement.

```
# Assign `import_file` to the name of the file

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.

remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]

# First line of `with` statement

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

Read the file contents

I believe that after opening the file, the .read() method is essential to convert the contents into a string. I believe that this string format allows me to later organize and extract IP addresses efficiently. I applied .read() to the file variable and stored the resulting string in the variable ip_addresses.

```
# Assign `import_file` to the name of the file

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.

remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]

# Build `with` statement to read in the initial contents of the file

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

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

  ip_addresses = file.read()

# Display `ip_addresses`

print(ip_addresses)
```

```
ip_address 192.168.205.12 192.168.6.9 192.168.52.90 192.168.90.124 192.168.186.176 192.168.133.188 192.168.218.219 192.168
.52.37 192.168.156.224 192.168.60.153 192.168.69.116
```

Convert the string into a list

I believe that to remove individual IP addresses, I need the allow list in list format. I used the .split() method to convert the ip_addresses string into a list. I believe that .split() breaks the string into a list by whitespace by default. This makes it easier to remove specific IP addresses from the allow list. I reassigned the resulting list back to the variable ip_addresses.

```
# Assign `import_file` to the name of the file

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.

remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]

# Build `with` statement to read in the initial contents of the file

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

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

  ip_addresses = file.read()

# Use `.split()` to convert `ip_addresses` from a string to a list

ip_addresses = ip_addresses.split()

# Display `ip_addresses`

print(ip_addresses)
```

```
['ip_address', '192.168.205.12', '192.168.6.9', '192.168.52.90', '192.168.90.124', '192.168.186.176', '192.168.133.188', '
192.168.218.219', '192.168.52.37', '192.168.156.224', '192.168.60.153', '192.168.69.116']
```

Iterate through the remove list

I believe that iterating through the IP addresses in remove_list is a key part of the algorithm. I used a for loop to apply code to each element in the sequence. I believe that the loop variable element represents each IP address in the remove list as it iterates, allowing me to check each one individually.

```
# Assign `import_file` to the name of the file

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.

remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]

# Build `with` statement to read in the initial contents of the file

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

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

  ip_addresses = file.read()

# Use `.split()` to convert `ip_addresses` from a string to a list

ip_addresses = ip_addresses.split()

# Build iterative statement
# Name loop variable `element`
# Loop through `ip_addresses`

for element in ip_addresses:

    # Display `element` in every iteration

    print(element)
```

```
ip_address
192.168.205.12
192.168.6.9
192.168.52.90
192.168.90.124
192.168.186.176
192.168.133.188
192.168.218.219
192.168.52.37
192.168.156.224
192.168.60.153
192.168.69.116
```

Remove IP addresses that are on the remove list

I believe that to remove IP addresses from the allow list, I first need a conditional that checks if element exists in ip_addresses. I believe this prevents errors from attempting to remove non-existent elements. I then applied the .remove() method to ip_addresses, passing in element as the argument, to ensure each IP address on the remove list is deleted from the allow list.

```
# Assign `import_file` to the name of the file

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.

remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]

# Build `with` statement to read in the initial contents of the file

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

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

  ip_addresses = file.read()

# Use `.split()` to convert `ip_addresses` from a string to a list

ip_addresses = ip_addresses.split()

# Build iterative statement
# Name loop variable `element`
# Loop through `ip_addresses`

for element in ip_addresses:

  # Build conditional statement
  # If current element is in `remove_list`,

    if element in remove_list:

        # then current element should be removed from `ip_addresses`

        ip_addresses.remove(element)

# Display `ip_addresses`

print(ip_addresses)
```

```
['ip_address', '192.168.205.12', '192.168.6.9', '192.168.52.90', '192.168.90.124', '192.168.186.176', '192.168.133.188', '
192.168.218.219', '192.168.52.37', '192.168.156.224', '192.168.60.153', '192.168.69.116']
```

Update the file with the revised list of IP addresses

I believe that after updating the list, I need to convert it back into a string using the .join() method. I believe that "\n".join(ip_addresses) ensures each IP address appears on a new line when written to the file. I then used another with statement along with .open() in write mode ("w") to update "allow_list.txt". I believe that using .write() on the file object allows me to overwrite the file with the updated IP addresses, ensuring that removed IPs no longer have access to the restricted content.

```
# Assign `import_file` to the name of the file

import_file = "allow_list.txt"

# Assign `remove_list` to a list of IP addresses that are no longer allowed to access restricted information.

remove_list = ["192.168.97.225", "192.168.158.170", "192.168.201.40", "192.168.58.57"]

# Build `with` statement to read in the initial contents of the file

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

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

  ip_addresses = file.read()

# Use `.split()` to convert `ip_addresses` from a string to a list

ip_addresses = ip_addresses.split()

# Build iterative statement
# Name loop variable `element`
# Loop through `ip_addresses`

for element in ip_addresses:

  # Build conditional statement
  # If current element is in `remove_list`,

    if element in remove_list:

        # then current element should be removed from `ip_addresses`

        ip_addresses.remove(element)

# Convert `ip_addresses` back to a string so that it can be written into the text file

ip_addresses = " ".join(ip_addresses)

# Build `with` statement to rewrite the original file

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

  # Rewrite the file, replacing its contents with `ip_addresses`

  file.write(ip_addresses)

# Build `with` statement to read in the updated file

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

    # Read in the updated file and store the contents in `text`

    text = file.read()

# Display the contents of `text`

print(text)
```

```
ip_address 192.168.205.12 192.168.6.9 192.168.52.90 192.168.90.124 192.168.186.176 192.168.133.188 192.168.218.219 192.168.52.37 19
2.168.156.224 192.168.60.153 192.168.69.116
```

```
# Define a function named `update_file` that takes in two parameters: `import_file` and `remove_list`
# and combines the steps you've written in this lab leading up to this

def update_file(import_file, remove_list):

  # Build `with` statement to read in the initial contents of the file

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

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

    ip_addresses = file.read()

  # Use `.split()` to convert `ip_addresses` from a string to a list

  ip_addresses = ip_addresses.split()

  # Build iterative statement
  # Name loop variable `element`
  # Loop through `ip_addresses`

  for element in ip_addresses:

    # Build conditional statement
    # If current element is in `remove_list`,

    if element in remove_list:

      # then current element should be removed from `ip_addresses`

      ip_addresses.remove(element)

  # Convert `ip_addresses` back to a string so that it can be written into the text file

  ip_addresses = " ".join(ip_addresses)

  # Build `with` statement to rewrite the original file

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

    # Rewrite the file, replacing its contents with `ip_addresses`

    file.write(ip_addresses)

# Call `update_file()` and pass in "allow_list.txt" and a list of IP addresses to be removed

update_file("allow_list.txt", ["192.168.25.60", "192.168.140.81", "192.168.203.198"])

# Build `with` statement to read in the updated file

with open("allow_list.txt", "r") as file:

  # Read in the updated file and store the contents in `text`

  text = file.read()

# Display the contents of `text`

print(text)
```

```
ip_address 192.168.205.12 192.168.6.9 192.168.52.90 192.168.90.124 192.168.186.176 192.168.133.188 192.168.218.219 192.168.52.37 19
2.168.156.224 192.168.60.153 192.168.69.116
```

Summary

I believe that I successfully created an algorithm that automates updating "allow_list.txt" by removing IP addresses in the remove_list. I believe the process of opening the file, reading it into a string, converting it into a list, iterating through elements, applying conditionals with .remove(), and finally converting the list back to a string to overwrite the file demonstrates effective use of Python for automation and security tasks.