Algorithm for file updates in Python

Project description

I am a security professional working at a health care company. 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 that must be removed from the allow list.

My 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, remove those IP addresses from the file containing the allow list.

Open the file that contains 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"]
# First line of `with` statement
with open(import_file, "r") as file:
```

In my algorithm, the with statement is used with the .open() function in read mode to open the allow list file for the purpose of reading it. The purpose of opening the file is to allow me to access the IP addresses stored in the allow list file. The with keyword will help manage the resources by closing the file after exiting the with statement.

Read the file contents

```
  ip_addresses = file.read()
# Display `ip_addresses`
print(ip_addresses)
```

I used file.read() to read the content of the file and using the print function displays the content of that 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
192.168.25.60
192.168.205.12
192.168.97.225
```

Convert the string into a list

```
# 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', '192.168.25.60', '192.168.205.12', '192.168.97.225', '192.168.6.9', '192.168.52.90', '192.168.158.170', '192.168.90.124', '192.168.186.176', '192.168.133.188', '192.168.203.198', '192.168.201.40', '192.168.218.219', '192.168.52.37', '192.168.156.224', '192.168.60.153', '192.168.58.57', '192.168.69.116']
```

Using the split() function it allows me to convert any string data into a list.

Iterate through the remove list

```
# 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
192.168.25.60
192.168.205.12
192.168.97.225
192.168.6.9
```

Using a for loop I am able to apply a specific code statements to all elements within a sequence.

Remove IP addresses that are on the remove list

```
  # 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', '192.168.25.60', '192.168.205.12', '192.168.6.9', '192.
8.133.188', '192.168.203.198', '192.168.218.219', '192.168.52.37', '19
```

Using the remove() function in my conditional statement I am able to remove all of the ip addressed that are no longer allowed to access the restricted data in the file.

Update the file with the revised list of IP addresses

```
# 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)
```

Summary

In summary, all the following steps provided in this document demonstrate how to use Python to import files, read and write files, and define functions to later be called. These steps are important for helping security professionals automate everyday tasks that would otherwise take much more time. Making tasks easier and quicker to do allows a security team to better protect company assets and customer data.