

Project Title: Automated IP Access Control Cleanup Using Python

Category: Blue Team Automation · Access Control Enforcement · File Manipulation

Tools/Tech: Python · Conditional Logic · List Filtering · Log Hygiene

Relevant MITRE ATT&CK Tactics:

- **TA0001 Initial Access**
- **T1078 Valid Accounts**
- **TA0005 Defense Evasion**

Problem Statement

In any security operations environment, keeping an allow list accurate and up to date is mission-critical. Whether you're managing firewall rules, cloud security groups, or application-level ACLs, stale IPs can lead to **unauthorized access**, **insider threats**, or even **compliance violations**.

In this project, I simulated an automated remediation task often handled by junior SOC analysts or security engineers: cleaning up a sensitive IP allow list by parsing a live file and **removing revoked IPs programmatically** using Python.

Technical Objective

Develop a Python-based algorithm to safely ingest an existing allow list of IP addresses, compare it to a known deny list (`remove_list`), strip out unauthorized entries, and update the original file — all without introducing errors or overwriting valid entries.

This was broken into seven operational tasks, each representing a discrete step in a typical SOC or SecOps workflow.

Task Breakdown & Execution

Task 1: Initial File Setup & Inspection

- **Objective:** Assign the allow list file name and verify its content structure.
- **Action:** Declared `import_file = "allow_list.txt"` and displayed the variable to confirm input validity.

Task 1

Your eventual goal is to develop an algorithm that parses a series of IP addresses that can access restricted information and removes the addresses that are no longer allowed. Python can automate this process.

You're given a text file called "allow_list.txt" that contains a series of IP addresses that are allowed to access restricted information.

There are IP addresses that should no longer have access to this information, and their IP addresses need to be removed from the text file. You're given a variable named `remove_list` that contains the list of IP addresses to be removed.

Display both variables to explore their contents, and run the cell. Be sure to replace each `### YOUR CODE HERE ###` with your own code before running the following cell.

```
In [1]: # 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"]

# Display 'import_file'
print(import_file)

# Display 'remove_list'
print(remove_list)
```

allow_list.txt

```
In [1]: # 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"]

# Display 'import_file'
print(import_file)

# Display 'remove_list'
print(remove_list)

allow_list.txt
['192.168.97.225', '192.168.158.170', '192.168.201.40', '192.168.58.57']
```

✔ Task 2-3: Reading and Loading File Contents

- **Objective:** Read the text file using the `with open()` context manager and store the content.
- **Result:** Loaded raw data into a string for further parsing.

Task 2

In this task, start by opening the text file using the `import_file` variable, the `with` keyword, and the `open()` function with the `"r"` parameter. Be sure to replace the `### YOUR CODE HERE ###` with your own code.

For now, you'll write the first line of the `with` statement. Running this code will produce an error because it will only contain the first line of the `with` statement; you'll complete this `with` statement in the task after this.

```
In [ ]: # 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:
```

Task 3

Now, use the `.read()` method to read the imported file and store it in a variable named `ip_addresses`.

Afterwards, display `ip_addresses` to examine the data in its current format.

Be sure to replace each `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
In [2]: # 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
192.168.25.60
192.168.205.12
```

```
    # 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
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
```

✓ Task 4: Convert String to List Format

- **Objective:** Convert the file string into a manipulable Python list using `.split()`.
- **Why:** Needed to iterate through each IP individually for comparisons and removals.

Task 4

After reading the file, reassign the `ip_addresses` variable so its data type is updated from a string to a list. Use the `.split()` method to achieve this. Adding this step will allow you to iterate through each of the IP addresses in the allow list instead of navigating a large string that contains all the addresses merged together.

Afterwards, display the `ip_addresses` variable to verify that the update took place.

Be sure to replace each `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
In [3]: # 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)
```

```
In [3]: # 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', '192.168.25.60', '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']
```

✓ Task 5-6: Filtering the Allow List

- **Objective:** Iterate over `ip_addresses` and remove all elements found in `remove_list`.
- **Challenge:** Avoid modifying the list while iterating — used slice-copy technique for safe mutation.

- This mimics a real-world **blacklist enforcement scenario**, where IPs flagged by threat intelligence (via feeds or alerts) need to be dynamically removed from access groups.

Task 5

Now, you'll write code that removes the elements of `remove_list` from the `ip_addresses` list. This will require both an iterative statement and a conditional statement.

First, build the iterative statement. Name the loop variable `element`, loop through `ip_addresses`, and display each element. Be sure to replace each `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
In [5]: # 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:
```

```
# 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
192.168.25.60
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
```

Task 6

Now, build a conditional statement to remove the elements of `remove_list` from the `ip_addresses` list. The conditional statement should be placed inside the iterative statement that loops through `ip_addresses`. In every iteration, if the current element in the `ip_addresses` list is in the `remove_list`, the `remove()` method should be used to remove that element.

Afterwards, display the updated `ip_addresses` list to verify that the elements of `remove_list` are no longer in the `ip_addresses`. Be sure to replace each `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
In [8]: # 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'
```

```

# 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', '192.168.25.60', '192.168.205.12', '192.168.6.9', '192.168.52.90', '192.168.90.124', '192.168.186.176', '192.168.133.188', '192.168.203.198', '192.168.218.219', '192.168.52.37', '192.168.156.224', '192.168.60.153', '192.168.69.116']

```

✓ Task 7: Overwriting the Cleaned Allow List

- **Objective:** Convert the cleaned list back into a string and write it to the same file.
- **Method:** Used " ".join() to reformat, then wrote to file using "w" mode for overwrite.

Task 7

The next step is to update the original file that was used to create the `ip_addresses` list. A line of code containing the `.join()` method has been added to the code so that the file can be updated. This is necessary because `ip_addresses` must be in string format when used inside the `with` statement to rewrite the file.

The `.join()` method takes in an iterable (such as a list) and concatenates every element of it into a string. The `.join()` method is applied to a string consisting of the character that will be used to separate every element in the iterable once its converted into a string. In the code below, the method is applied to the string `" "`, which contains just a space character. The argument of the `.join()` method is the iterable you want to convert, and in this case, that's `ip_addresses`. As a result, it converts `ip_addresses` from a list back into a string with a space between each element and the next.

After this line with the `.join()` method, build the `with` statement that rewrites the original file. Use the `"w"` parameter when calling the `open()` function to delete the contents in the original file and replace it with what you want to write. Be sure to replace each `### YOUR CODE HERE ###` with your own code before you run the following cell. This code cell will not produce an output.

```
In [10]: # 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)
```






Final Outcome

- ☒ Fully automated the removal of unauthorized IPs from a critical allow list.
- ☒ Demonstrated safe read-modify-write cycles for sensitive files — preventing data loss.
- ☒ Built a Python-based remediation script that mirrors a lightweight **playbook task** you'd find in a SOAR or SIEM-integrated workflow.
- ☒ Developed core logic that can be extended to:
 - SIEM-driven scripts that pull from detection logs
 - Threat intel-based feed filtering

- Compliance checks for stale entries in privileged access groups

Transferable Knowledge & SOC Readiness

This project may look deceptively simple, but it showcases real, **production-relevant thinking**:

-  **Data Integrity:** File operations were handled with care to avoid accidental truncation or write errors — a must for log handling.
-  **Automation Mindset:** Reduced manual overhead, minimized human error, and enabled fast iteration over access policy enforcement.
-  **Extensibility:** This logic could be ported into a larger SIEM pipeline, a bash automation script, or a Python-based security orchestration tool.

Final Thoughts

This project wasn't about flashy tooling — it was about **discipline, clarity, and control**. I treated every IP like a real access key and every file like a production ACL. That's the mindset I bring to blue team operations: precision, automation, and clean policy enforcement.