📁 Portfolio Entry: Log File Forensics and IP Access Control Automation with Python

🔍 Project: "Turning Raw Logs into Actionable Security Data"

Tools/Tech: Python · File I/O · String Parsing · Access Lists · Logging · Conditional Logic
Skills Demonstrated: Log handling · Input/output sanitization · Data appending · Access control structuring · Automated script logic

---

🧩 Problem Overview

In real-world SOC operations, raw logs and access control files are often messy, incomplete, or improperly maintained. In this project, I simulated the start-to-finish process of ingesting and preparing a **security log file**, parsing and cleaning that data, correcting missing entries, and constructing a new **allow list** of IP addresses — all using Python.

The tasks mirror real-life workflows a SOC analyst might face during the early stages of log ingestion, detection engineering, or threat triage scripting.

⚙️ Task Breakdown & Hands-On Execution

✅ Task 1: Importing a Security Log File

I began by using Python's `with open()` syntax to safely import a raw `.txt` security log file. This models a standard log ingestion operation — the first stage before parsing alerts or identifying anomalies.

Task 1

In this task, you'll import a security log text file and store it as a string to prepare it for analysis.

In Python, a `with` statement is often used in file handling to open a file and then automatically close the file after reading it.

You're given a variable named `import_file` that contains the name of the log file that you want to import. Start by writing the first line of the `with` statement in the following code cell. Use the `open()` function, setting the second parameter to `"r"`. Note that 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. Be sure to replace the `### YOUR CODE HERE ###` with your own code.

In [2]:
```
# Assign `import_file` to the name of the text file that contains the security log file

import_file = "login.txt"

# First line of the `with` statement
# Use `open()` to import security log file and store it as a string

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

✅ Task 2: Reading and Displaying Log Data

I applied `.read()` to store the full file contents in a `text` variable, then printed it to review structure and identify formatting issues. This mirrors a **triage phase** where analysts manually review raw logs before writing parsers or log source normalization rules (e.g. in Elastic or Splunk).

Task 2

Now, you'll use the `.read()` method to read the imported file, and you'll store the result in a variable named `text`. Afterwards, display the `text` and explore what it contains by running the cell. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

In [3]:
```
# Assign `import_file` to the name of the text file that contains the security log file

import_file = "login.txt"

# The `with` statement
# Use `open()` to import security log file and store it as a string

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

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

  text = file.read()

# Display the contents of `text`

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
jlansky,192.168.238.42,1:07:11,2022-05-11
acook,192.168.52.90,9:56:48,2022-05-10
asundara,192.168.58.217,23:17:52,2022-05-12
jclark,192.168.214.49,20:49:00,2022-05-10
```

✅ Task 3: Splitting Logs into Lines

Next, I split the entire log file into a list — one entry per line — using `.split("\n")`. This step was critical for later automation (filtering, searching, appending). It reflects how log pipelines break up multiline data into processable formats for SIEM alerting.

Task 3

The output in the previous step is one big string. In this task, you'll explore how you can split the string that contains the entire imported log file into a list of strings, one string per line.

Use the `.split()` method to perform this split and then display the result. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

Note that displaying `.split()` doesn't change what is stored in the `text` variable. Variable reassignment would be necessary if you want to store the result after splitting.

In [6]:
```
# Assign `import_file` to the name of the text file that contains the security log file

import_file = "login.txt"

# The `with` statement
# Use `open()` to import security log file and store it as a string

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

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

  text = file.read()

# Display the contents of `text` split into separate lines

print(text.split("\n"))
```

```
['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', 'jlansky,192.168.238.42,1:07:11,2022-05-11', 'acook,192.168.52.90,9:56:48,2022-05-10', 'asundara,192.168.58.217,23:17:52,2022-05-12', 'jclark,192.168.214.49,20:49:00,2022-05-10', 'cjackson,192.168.247.153,19:36:42,2022-05-12', 'jclark,192.168.197.247,14:11:04,2022-05-12', 'apatel,192.168.46.207,17:39:42,2022-05-
```

In [6]:
```
# Assign `import_file` to the name of the text file that contains the security log file

import_file = "login.txt"

# The `with` statement
# Use `open()` to import security log file and store it as a string

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

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

  text = file.read()

# Display the contents of `text` split into separate lines

print(text.split("\n"))
```

```
['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', 'jlansky,192.168.238.42,1:07:11,2022-05-11', 'acook,192.168.52.90,9:56:48,2022-05-10', 'asundara,192.168.58.217,23:17:52,2022-05-12', 'jclark,192.168.214.49,20:49:00,2022-05-10', 'cjackson,192.168.247.153,19:36:42,2022-05-12', 'jclark,192.168.197.247,14:11:04,2022-05-12', 'apatel,192.168.46.207,17:39:42,2022-05-10', 'mabadi,192.168.96.244,10:24:43,2022-05-12', 'iuduike,192.168.131.147,17:50:00,2022-05-11', 'abellmas,192.168.60.111,13:37:05,2022-05-10', 'gesparza,192.168.148.80,6:30:14,2022-05-11', 'cgriffin,192.168.4.157,23:04:05,2022-05-09', 'alevitsk,192.168.210.228,8:10:43,2022-05-08', 'eraab,192.168.24.12,11:29:27,2022-05-11', 'jsoto,192.168.25.60,5:09:21,2022-05-09', '']
```

```
[7]: # Assign `import_file` to the name of the text file that contains the security log file

     import_file = "login.txt"

     # The `with` statement
     # Use `open()` to import security log file and store it as a string

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

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

       text = file.read()

     # Display the contents of `text` split into separate lines
     lines = [line for line in text.split("\n") if line.strip() != ""]
     header = lines[0].split(",")
     data = [dict(zip(header, row.split(","))) for row in lines[1:]]

     print(data)
```

```
[{'username': 'tshah', 'ip_address': '192.168.92.147', 'time': '15:26:08', 'date': '2022-05-10'}, {'username': 'dta
naka', 'ip_address': '192.168.98.221', 'time': '9:45:18', 'date': '2022-05-09'}, {'username': 'tmitchel', 'ip_addre
ss': '192.168.110.131', 'time': '14:13:41', 'date': '2022-05-11'}, {'username': 'daquino', 'ip_address': '192.168.1
68.144', 'time': '7:02:35', 'date': '2022-05-08'}, {'username': 'eraab', 'ip_address': '192.168.170.243', 'time':
'1:45:14', 'date': '2022-05-11'}, {'username': 'jlansky', 'ip_address': '192.168.238.42', 'time': '1:07:11', 'dat
e': '2022-05-11'}, {'username': 'acook', 'ip_address': '192.168.52.90', 'time': '9:56:48', 'date': '2022-05-10'},
{'username': 'asundara', 'ip_address': '192.168.58.217', 'time': '23:17:52', 'date': '2022-05-12'}, {'username': 'j
clark', 'ip_address': '192.168.214.49', 'time': '20:49:00', 'date': '2022-05-10'}, {'username': 'cjackson', 'ip_add
ress': '192.168.247.153', 'time': '19:36:42', 'date': '2022-05-12'}, {'username': 'jclark', 'ip_address': '192.168.
197.247', 'time': '14:11:04', 'date': '2022-05-12'}, {'username': 'apatel', 'ip_address': '192.168.46.207', 'time':
'17:39:42', 'date': '2022-05-10'}, {'username': 'mabadi', 'ip_address': '192.168.96.244', 'time': '10:24:43', 'dat
```

```
In [10]: # Assign `import_file` to the name of the text file that contains the security log file
         import_file = "login.txt"

         # Open and read the file
         with open(import_file, "r") as file:
             text = file.read()

         # Step 1: Clean and split into lines
         lines = [line for line in text.strip().split("\n") if line.strip() != ""]

         # Step 2: Extract header and data
         header = lines[0].split(",")  # ['username', 'ip_address', 'time', 'date']
         rows = lines[1:]  # skip the header

         # Step 3: Convert each row to a dictionary
         log_data = [dict(zip(header, row.split(","))) for row in rows]

         # Step 4: Print all usernames
         for entry in log_data:
             print(entry["username"])
```

```
tshah
dtanaka
tmitchel
daquino
eraab
jlansky
acook
asundara
jclark
cjackson
jclark
apatel
```

✅ Task 4: Appending a Missing Log Entry

After detecting a missing log event (simulating data loss during log rotation or source failure), I restored integrity by appending the missing log string to the end of the file using the `"a"` (append) mode. Then I verified success by re-reading the file.

Task 4

There is a missing entry in the log file. You'll need to account for that by appending it to the log file. You're given the missing entry stored in a variable named `missing_entry`.

Use the `.write()` method and the parameter `"a"` in the `open()` function. Be sure to replace each `### YOUR CODE HERE ###` with your own code before you run the following cell.

After the portion of the code that writes to the file, another with statement uses the `.read()` method to read the updated file into the `text` variable and then display it.

In [12]:
```
# Assign `import_file` to the name of the text file that contains the security log file

import_file = "login.txt"

# Assign `missing entry` to a log that was not recorded in the log file

missing_entry = "jrafael,192.168.243.140,4:56:27,2022-05-09"

# Use `open()` to import security log file and store it as a string
# Pass in "a" as the second parameter to indicate that the file is being opened for appending purposes

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

    # Use `.write()` to append `missing_entry` to the log file

   text = file.write(missing_entry)

# Use `open()` with the parameter "r" to open the security log file for reading purposes

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

    # Use `.read()` to read in the contents of the log file and store in a variable named `text`
```

```
# Use `open()` with the parameter "r" to open the security log file for reading purposes

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

    # Use `.read()` to read in the contents of the log file and store in a variable named `text`

    text = file.read()

# Display the contents of `text`

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
jlansky,192.168.238.42,1:07:11,2022-05-11
acook,192.168.52.90,9:56:48,2022-05-10
asundara,192.168.58.217,23:17:52,2022-05-12
jclark,192.168.214.49,20:49:00,2022-05-10
cjackson,192.168.247.153,19:36:42,2022-05-12
jclark,192.168.197.247,14:11:04,2022-05-12
apatel,192.168.46.207,17:39:42,2022-05-10
mabadi,192.168.96.244,10:24:43,2022-05-12
iuduike,192.168.131.147,17:50:00,2022-05-11
abellmas,192.168.60.111,13:37:05,2022-05-10
gesparza,192.168.148.80,6:30:14,2022-05-11
cgriffin,192.168.4.157,23:04:05,2022-05-09
alevitsk,192.168.210.228,8:10:43,2022-05-08
eraab,192.168.24.12,11:29:27,2022-05-11
jsoto,192.168.25.60,5:09:21,2022-05-09
jrafael,192.168.243.140,4:56:27,2022-05-09
```

✅ Task 5: Creating an IP Allow List

I created a new file `allow_list.txt` and used a variable `ip_addresses` containing a string of trusted IPs. This reflects a foundational part of **access control and segmentation enforcement**.

Task 5

The next task you're responsible for is creating a text file. This text file should include a list of IP addresses that are allowed to access restricted information. Documenting this in a text file will help you communicate your findings to your security team.

Start by creating a variable named `import_file` that stores the name of the file, which should be `"allow_list.txt"`.

You're also given a variable named `ip_addresses` that stores a string containing the IP addresses that are allowed.

Run the code to display the two variables and explore what they contain. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
[14]: # Assign `import_file` to the name of the text file that you want to create

import_file = "allow_list.txt"

# Assign `ip_addresses` to a list of IP addresses that are allowed to access the restricted information

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.

# Display `import_file`

print(import_file)

# Display `ip_addresses`
```

```
allow_list.txt
192.168.218.160 192.168.97.225 192.168.145.158 192.168.108.13 192.168.60.153 192.168.96.200 192.168.247.153 192.16
8.3.252 192.168.116.187 192.168.15.110 192.168.39.246
```

✅ Task 6: Writing to the Allow List File

Using `"w"` mode, I wrote the trusted IPs to the file — ensuring it could be used by firewall scripts, ACL updates, or monitoring tools.

Task 6

Your next goal is to create a `with` statement in order to write the IP addresses to the text file you created in the previous step.

You'll first open the file using the `"w"` parameter. Then, you'll write the IP addresses to the file. Be sure to replace each `### YOUR CODE HERE ###` with your own code before you run the following cell. Note that the code cell will contain a `with` statement that writes to a file but does not display information to the screen, so running it will not produce an output.

```
In [ ]: # Assign `import_file` to the name of the text file that you want to create

import_file = "allow_list.txt"

# Assign `ip_addresses` to a list of IP addresses that are allowed to access the restricted information

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.

# Create a `with` statement to write to the text file

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

  # Write `ip_addresses` to the text file
```

✅ Task 7: Reading the Final IP Allow List

To confirm the success of the write operation, I used a final `with open(..., "r")` statement to read and print the contents.

Task 7

In this final step, you'll complete the code you've been writing up to this point. You'll add code to read the file containing IP addresses.

Complete a `with` statement that reads the text file and stores it in a new variable called `text`.

Afterwards, display the contents of `text` and run the cell to explore the result. Be sure to replace each `### YOUR CODE HERE ###` with your own code before you run the following cell.

In [15]:
```
# Assign `import_file` to the name of the text file that you want to create

import_file = "allow_list.txt"

# Assign `ip_addresses` to a list of IP addresses that are allowed to access the restricted information

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.

# Create a `with` statement to write to the text file

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

    # Write `ip_addresses` to the text file

    file.write(ip_addresses)

# Create a `with` statement to read in the text file

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

    # Read the file and store the result in a variable named `text`

    text = file.read()
```

In [15]:
```
# Assign `import_file` to the name of the text file that you want to create

import_file = "allow_list.txt"

# Assign `ip_addresses` to a list of IP addresses that are allowed to access the restricted information

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.

# Create a `with` statement to write to the text file

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

    # Write `ip_addresses` to the text file

    file.write(ip_addresses)

# Create a `with` statement to read in the text file

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

    # Read the file and store the result in a variable named `text`

    text = file.read()

# Display the contents of `text`
```

```
192.168.218.160 192.168.97.225 192.168.145.158 192.168.108.13 192.168.60.153 192.168.96.200 192.168.247.153 192.16
8.3.252 192.168.116.187 192.168.15.110 192.168.39.246
```

🔍 Final Results & Reflections

Across this project, I performed a full cycle of log handling and access control enforcement using Python — replicating the kind of scripting analysts use to **triage, enrich, and automate security data workflows.**

📈 **Outcomes:**

- Parsed raw logs for downstream analysis
- Repaired broken or missing data records
- Built clean, reusable access control files
- Strengthened practical Python skills in file I/O, string handling, and basic automation

🛡️ **SOC Relevance:**
This project reflects the kind of hands-on technical triage done in modern blue teams — especially in environments using:

- **SIEMs** like Splunk, ELK, or Chronicle
- **Detection rules** that depend on preprocessed logs
- **IP reputation feeds and allow/block lists** used in NIDS/HIDS or EDR tooling
- **MITRE ATT&CK mappings**, particularly under:
  - `T1078` (Valid Accounts)
  - `T1036` (Masquerading)
  - `T1566` (Initial Access via phishing, from external IPs)

🚀 Why This Project Matters

It's easy to overlook file-handling as "basic," but in cybersecurity, broken logs and bad data are where attacks hide. This project shows I understand the full chain: **from ingestion to enrichment to enforcement.**

I didn't just write Python — I treated the log file like real evidence, the IP list like real policy, and the automation like real prevention. That's the mindset I bring to any SOC team.