


Project: Automating User Alerts and Username Aggregation with Python Functions

 Role: Security Analyst (Blue Team – Automation Focus)

 Focus Area: Alerting Logic, Function Design, and Username Parsing

 Tools/Concepts: Python Functions, Loops, String Concatenation, Output Formatting

Overview: Automating Security Alerts and User String Processing

In high-volume SOC environments, **automation is key** — not just for detection, but for communication. This project simulates the process of building and refining Python functions to automate:

- Triggering user alerts at scale
 - Converting structured data (lists of usernames) into readable, actionable output
 - Writing reusable logic that mimics common **log enrichment** and **SIEM data formatting** patterns
-

Task Breakdown

Task 2–3: Alert Function with Iterative Output

Objective:

Demonstrate how Python functions can automate repetitive alerting behaviors.

Implementation:

- **Task 2:** Called a pre-defined alert function to analyze the default output — simulating a one-time alert message.

- **Task 3:** Worked with an enhanced version that used a **for loop** to repeat alert messages — a crucial pattern for simulating persistent event detection or recurring issues (e.g., repeated failed logins).

SOC Relevance:

This kind of repetitive alerting is seen in:

- **Brute-force detection scripts**
- Custom playbook triggers inside tools like **Splunk SOAR**, **Cortex XSOAR**, or **SIEM correlation rules**

[Double-click to edit this markdown cell and write something here.]

Task 2

For this task, call the `alert()` function that was defined earlier and analyze the output.

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

```

In [1]: # Define a function named 'alert()'

def alert():
    print("Potential security issue. Investigate further.")

# Call the 'alert()' function
alert()

Potential security issue. Investigate further.

```

Task 3

Functions can include other components that you've already worked with. The following code cell contains a variation of the `alert()` function that now uses a `for` loop to display the alert message multiple times.

For this task, call the new `alert()` function and observe the output.

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

```

In [2]: # Define a function named 'alert()'

def alert():
    for i in range(3):
        print("Potential security issue. Investigate further.")

# Call the 'alert()' function
alert()

Potential security issue. Investigate further.
Potential security issue. Investigate further.
Potential security issue. Investigate further.

```

Task 4–5: Building and Testing the `list_to_string()` Function

Objective:

Create a custom Python function that receives a list of usernames and outputs each name clearly, line by line.

Implementation:

- **Task 4:** Defined a reusable function `list_to_string()`
- **Task 5:** Used a `for` loop to iterate through `username_list` and print each element individually.

SOC Use Case:

Functions like this are directly useful in:

- Parsing usernames from **SIEM query results**
- Reporting active directory accounts with expired credentials
- Alerting analysts to suspicious login activity across multiple hosts

Task 4

In the next part of your work, you're going to work with a list of approved usernames, representing users who can enter a system. You'll be developing a function that helps you convert the list of approved usernames into one big string. Structuring this data differently enables you to work with it in different ways. For example, structuring the usernames as a list allows you to easily add or remove a username from it. In contrast, structuring it as a string allows you to easily place its contents into a text file.

For this task, start defining a function named `list_to_string()`. Write the function header.

Be sure to replace the `### YOUR CODE HERE ###` with your own code. Note that running this cell will produce an error since this cell will just contain the function header; you'll write the function body and complete the function definition in a later task.

```
In [ ]: # Define a function named 'list_to_string()'

def list_to_string(user_list):
```

Task 5

Now you'll begin to develop the body of the `list_to_string()` function.

In the following code cell, you're provided a list of approved usernames, stored in a variable named `username_list`. Your task is to complete the body of the `list_to_string()` function. Recall that the body of a function must be indented. To complete the function body, write a loop that iterates through the elements of the `username_list` and displays each element. Then, call the function and run the cell to observe what happens.

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

```
In [5]: # Define a function named 'list_to_string()'

def list_to_string():

    # Store the list of approved usernames in a variable named 'username_list'
    username_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab", "gesparza", "alevitsk", "wjaffrey"]

    # Write a for loop that iterates through the elements of 'username_list' and displays each element
    for i in username_list:
        print(i)

    # Call the 'list_to_string()' function
    list_to_string()

elarson
bmoreno
tshah
sgilmore
eraab
gesparza
```

Task 6–7: String Concatenation for Readable Username Logs

Objective:

Enhance the `list_to_string()` function to return usernames as a single, clean string — ideal for SIEM dashboards, alerts, or reports.

Implementation:

- **Task 6:** Used string concatenation in a loop to combine all usernames into one output variable.
- **Task 7:** Added ", " between usernames to improve readability, reflecting best practices in log formatting and alert summaries.

python

```
sum_variable = ""
for username in username_list:
    sum_variable += username + ", "
print(sum_variable)
```

SOC Relevance:

This task mimics what you'd do when:

- Formatting user activity for a **SIEM alert description**
- Generating custom alert outputs or Slack/Teams notifications
- Structuring human-readable logs before sending to ticketing systems like **ServiceNow** or **Jira**

Task 6

String concatenation is a powerful concept in coding. It allows you to combine multiple strings together to form one large string, using the addition operator (+). Sometimes analysts need to merge individual pieces of data into a single string value. In this task, you'll use string concatenation to modify how the `list_to_string()` function is defined.

In the following code cell, you're provided a variable named `sum_variable` that initially contains an empty string. Your task is to use string concatenation to combine the usernames from the `username_list` and store the result in `sum_variable`.

In each iteration of the `for` loop, add the current element of `username_list` to `sum_variable`. At the end of the function definition, write a `print()` statement to display the value of `sum_variable` at that stage of the process. Then, run the cell to call the `list_to_string()` function and examine its output.

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

```
In [7]: # Define a function named `list_to_string()`
def list_to_string():
    # Store the list of approved usernames in a variable named `username_list`
    username_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab", "gesparza", "alevitsk", "wjaffrey"]
    # Assign `sum_variable` to an empty string
    sum_variable = ""
    # Write a for loop that iterates through the elements of `username_list` and displays each element
```

and examine its output.

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

```
In [7]: # Define a function named `list_to_string()`
def list_to_string():
    # Store the list of approved usernames in a variable named `username_list`
    username_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab", "gesparza", "alevitsk", "wjaffrey"]
    # Assign `sum_variable` to an empty string
    sum_variable = ""
    # Write a for loop that iterates through the elements of `username_list` and displays each element
    for i in username_list:
        sum_variable = sum_variable + i
    # Display the value of `sum_variable`
    print(sum_variable)
# Call the `list_to_string()` function
list_to_string()

elarsonbmorenotshahsgilmoreeraabgesparzaalevitskwjaffrey
```

Task 7

In this final task, you'll modify the code you wrote previously to improve the readability of the output.

This time, in the definition of the `list_to_string()` function, add a comma and a space (", ") after each username. This will prevent all the usernames from running into each other in the output. Adding a comma helps clearly separate one username from the next in the output. Adding a space following the comma as an additional separator between one username and the next makes it easier to read the output. Then, call the function and run the cell to observe the output.

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

```
In [8]: # Define a function named `list_to_string()`

def list_to_string():

    # Store the list of approved usernames in a variable named `username_list`

    username_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab", "gesparza", "alevitsk", "wjaffrey"]

    # Assign `sum_variable` to an empty string

    sum_variable = ""

    # Write a for loop that iterates through the elements of `username_list` and displays each element

    for i in username_list:
        sum_variable = sum_variable + i + ", "

    # Display the value of `sum_variable`




    print(sum_variable)

# Call the `list_to_string()` function

list_to_string()

elarson, bmoreno, tshah, sgilmore, eraab, gesparza, alevitsk, wjaffrey,
```

✓ Key Takeaways

Skill	Real-World Application
 Looping & String Concatenation	Alert enrichment, custom report formatting
 Function Modularity	SOC playbook design, reusable automation
 Data Structuring & Readability	Analyst communication, ticket quality, SOC escalation messages

Skill

Real-World Application



SOC Mindset

Think like a responder: readable, actionable output over raw data dumps



Final Thoughts

This project might look simple — but behind it is **SOC-grade logic**:

- Turning scattered user data into readable formats
- Writing clean functions that scale well across incident types
- Following the same design logic used in custom playbooks, alert scripts, and log normalization filters

In a SOC, clarity isn't optional. This project proves I know how to turn raw data into clean, actionable insight — fast.