Activity - Creating and Applying Conditional Statements for Security Automation

Introduction

This project demonstrates the use of conditional statements in Python to automate common security analyst tasks, such as verifying operating system updates and evaluating login attempts based on user approval and timing.

Conditional statements are vital for automating decisions that depend on specific conditions and streamlining processes in cybersecurity workflows.

Scenario

As a security analyst:

1. Check if a user's operating system requires an update.
2. Evaluate login attempts to determine:
    - If the user has permission to access the device.
    - If the login occurred during organization hours.

Tasks and Solutions

Task 1: Verify OS Update Requirements

Objective: Automate OS update checks for three operating systems:

- OS 1 and OS 3 require updates.
- OS 2 is up-to-date.

python

```
system = "OS 2"

if system == "OS 2":
   print("no update needed")
```

Outcome:

- If the system is set to "OS 2", display "no update needed."
- For other OS values, no message is displayed.

Task 2: Add Update Notifications for Other OS

Objective: Notify users if their OS requires an update.

python

```
system = "OS 3"

if system == "OS 2":
   print("no update needed")
else:
   print("update needed")
```

Outcome:

- "update needed" displays for OS 1 and OS 3.
- "no update needed" displays for OS 2.

Task 3: Handle Undefined OS Inputs

Objective: Improve feedback for invalid OS inputs.

python

```
system = "OS 4"

if system == "OS 2":
    print("no update needed")
elif system == "OS 1" or system == "OS 3":
    print("update needed")
```

Outcome:

- Valid inputs (OS 1, OS 2, OS 3) return correct update messages.
- Invalid inputs like "OS 4" result in no message.

Task 4: Simplify Conditions

Objective: Use logical operators for concise conditionals.

python

```
system = "OS 4"

if system == "OS 2":
    print("no update needed")
elif system in ["OS 1", "OS 3"]:
    print("update needed")
```

Outcome: Simplifies logic using a list for valid OS checks.

Task 5: Evaluate Login Attempt by Username

Objective: Determine access based on pre-approved usernames.

python

```
approved_user1 = "elarson"
approved_user2 = "bmoreno"
username = "bmoreno"
```

```
if username == approved_user1 or username == approved_user2:
    print("This user has access to this device.")
else:
    print("This user does not have access to this device.")
```

Outcome: Approved users receive access confirmation; others are denied.

Task 6: Expand Username Validation

Objective: Use a list for easier management of approved usernames.

python

```
approved_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab"]
username = "jhill"

if username in approved_list:
    print("This user has access to this device.")
else:
    print("This user does not have access to this device.")
```

Outcome: The list simplifies validation for multiple users.

Task 7: Include Login Time Validation

Objective: Verify if the login occurred during organization hours.

python

```
organization_hours = True

if organization_hours:
    print("Login attempt made during organization hours.")
else:
    print("Login attempt made outside of organization hours.")
```

Outcome: Displays whether the login was during or outside organization hours.

Task 8: Combine Conditions

Objective: Check username and organization hours simultaneously.

python

```
approved_list = ["elarson", "bmoreno", "tshah", "sgilmore", "eraab"]
username = "bmoreno"
organization_hours = True

if username in approved_list and organization_hours:
    print("Login attempt made by an approved user during organization hours.")
else:
    print("Username not approved or login attempt made outside of organization hours.")
```

Outcome: A single conditional determines both user approval and login timing.

Summary

This project demonstrates the use of Python conditional statements for automating key security analyst tasks, including verifying operating system updates and validating login attempts based on user approval and timing. By leveraging comparison operators, logical operators, and lists, the solutions are both efficient and scalable.

Key Highlights:

1. **Automated OS Update Checks:** These checks identify whether a system is up-to-date or requires an update based on predefined conditions.
2. **User Access Validation:** This process confirms whether a username is authorized to access a device using simple or list-based checks.
3. **Login Timing Verification:** Determines if login attempts occurred within organization hours.
4. **Combined Conditions:** A streamlined solution to check both user approval and timing in a single step.

Key Takeaways:

- Conditional logic is essential for security automation, enabling efficient and reliable decision-making.
- Logical operators (`and`, `or`, `in`) simplify complex condition checks.
- Writing readable and concise code enhances maintainability and scalability.