

Project: Automating OS Update Detection with Conditional Logic in Python

Problem Overview


In a real-world security operations environment, ensuring that endpoints are running up-to-date operating systems is **mission-critical**. Outdated systems often contain unpatched vulnerabilities that attackers can exploit using publicly known exploits (e.g., CVEs tied to OS version flaws).

In this project, I simulated a lightweight automation script designed to help security teams **quickly identify whether a user's system is up-to-date**, using simple Python logic. While the example is stripped down to 3 OS types, the logic can easily be extended into real-world vulnerability management use cases and SIEM rule engineering.

Technical Walkthrough

Task 1: Initial Conditional Check

- Goal: Check whether the user's system (OS 1, OS 2, or OS 3) is running the secure and updated version (OS 2).
- Solution: Used a Python `if` statement to compare the current system and display an update status message.

 **Decision Rationale:** Mirrors how a SOC might use endpoint data (e.g., from CrowdStrike, SCCM, or Nessus) to write conditional rules based on OS versioning.

Task 1

You are asked to help automate the process of checking whether a user's operating system requires an update. Imagine that a user's device can be running one of the following operating systems: OS 1, OS 2, or OS 3. While OS 2 is up-to-date, OS 1 and OS 3 are not. Your task is to check whether the user's system is up-to-date, and if it is, display a message accordingly. To do this, complete the conditional statement using the keyword `if`. Be sure to replace the `### YOUR CODE HERE ###` with your own code before you run the following cell.

```
5]: # Assign a variable named `system` to a specific operating system, represented as a string
# This variable indicates which operating system is running
# Feel free to run this cell multiple times; each time try assigning `system` to different values ("OS 1", "OS 2", "
system = "OS 2"


# If OS 2 is running, then display a "no update needed" message

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

no update needed
```

Task 2: System State Testing

- Modified the `system` variable to simulate each OS scenario and validated the logic.
- Tested for expected output messages when the OS is current (OS 2) vs. outdated (OS 1 / OS 3).

 **Testing Mindset:** Treated each test like validating a detection rule — checking for true/false conditions and refining response outputs.

Task 2

Now try assigning the `system` variable to different values ("OS 1" , "OS 2" , and "OS 3"), run the cell, and observe what happens. Keep the conditional statement as is. Be sure to replace the `### YOUR CODE HERE ###` with your own code.

```
7]: # Assign `system` to a specific operating system
# This variable represents which operating system is running
# Feel free to run this cell multiple times; each time try assigning `system` to different values ("OS 1", "OS 2", "
system = "OS 1"

# If OS 2 is running, then display a "no update needed" message

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

Task 2

Now try assigning the `system` variable to different values ("OS 1" , "OS 2" , and "OS 3"), run the cell, and observe what happens. Keep the conditional statement as is. Be sure to replace the `### YOUR CODE HERE ###` with your own code.

```
8]: # Assign `system` to a specific operating system
# This variable represents which operating system is running
# Feel free to run this cell multiple times; each time try assigning `system` to different values ("OS 1", "OS 2", "
system = "OS 2"

# If OS 2 is running, then display a "no update needed" message

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

no update needed
```

⚠️ Task 3: Adding Fallback Alerts

- Identified a logic gap: when the OS wasn't OS 2, the script showed no response.
- Solution: Added an `else` statement to provide a clear message if the OS requires an update.

🧠 **Transferable Skill:** This mirrors how SOC analysts or engineers refine detection rules — ensuring every potential input has a response or escalation path.

```
task 3

Nothing is displayed when the system is not equal to "OS 2". This is because the condition didn't evaluate to True .

It would be beneficial if an alternative message is provided to them when updates are needed.

In the following cell, add the appropriate keyword after the first conditional so that it will display a message that conveys that an update is needed when
the system is not running OS 2. Be sure to replace each ### YOUR CODE HERE ### with your own code.

Then, set the value of the system variable to indicate that OS 2 is running and run the cell. After observing what happens, set the value of system to
indicate either that OS 1 is running or that OS 3 is running and run the cell.

In [9]: # Assign 'system' to a specific operating system
# This variable represents which operating system is running

system = "OS 2"

# If OS 2 is running, then display a "no update needed" message
# Otherwise, display a "update needed" message

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

no update needed
```

🔄 Real-World Tie-In: Vulnerability Management Context

This exercise maps closely to basic logic used in:

- 🔍 **Nessus scans:** Checking OS fingerprints and comparing against patch levels.
- 🇮🇹 **SIEM correlation rules:** Creating logic that alerts when outdated systems connect to the network.
- 📦 **MITRE ATT&CK Technique T1203:** Exploitation for client execution often targets outdated OS-level vulnerabilities.

🎯 Reflection & Value

This simple script reflects the mindset of a SOC analyst who isn't just checking boxes but thinking **critically about threat exposure, logic completeness, and scalability**. Though basic, the structure supports real-world applications like:

- Automating ticket creation when outdated OS is detected
- Enforcing conditional logic inside SOAR playbooks
- Developing custom Splunk KQL or Sentinel analytic rules

Skills Demonstrated

Area	Description
Python Scripting	Used <code>if/else</code> logic to handle OS comparison cases
Security Automation	Designed logic that can scale into detection and patch compliance
Problem Solving	Improved incomplete logic to ensure no blind spots in detection
SOC Relevance	Mirrors how endpoint data gets parsed and evaluated in real-time