Part 1: Preparation Phase – Asset Inventory Script

Scenario: You are part of an IR team responsible for ensuring that all systems are accounted for and up to date before an incident occurs.

Task

1. Write a **PowerShell** or **Python** or **Bash** or **Batch** script to:
    - List all active users on a system.
    - Identify installed software and versions.
    - Check for missing security patches.
    - List auto runs
    - Identify USB History
2. Run the script on a test machine and record its output.

Part 2: Detection Phase – Suspicious Login Monitoring

Scenario: Your organization has noticed an increase in unauthorized access attempts. Your IR team must develop a script to monitor failed login attempts and detect potential brute-force attacks.

Task

1. Write a **Python** or **PowerShell** or **Bash** or **Batch** script to:
    - Parse system logs for failed login attempts.
    - Identify repeated login failures from the same IP.
    - Alert the user if a threshold (e.g., 5 failed attempts in 10 minutes) is exceeded.
    - Monitor privileged account Logins
2. Simulate failed logins and observe how the script detects them.

Part 3: Lab Report Submission (see "goby" outline appendix C)

After completing both tasks, students will submit a report that includes:
✅ The scripts used for both Preparation and Detection.
✅ A summary of findings (e.g., what the scripts revealed about the system).
✅ Reflections on the benefits of automation in Incident Response.

Your report should include:

- **Scripts:** The Script Source (PowerShell, Python, Bash, Batch scripts) you created (as above **or adapted**).
- **Findings:**
    - List of active users on the test system.
    - List of installed software and their versions.

    - Information about any missing security patches.
    - Output from the script showing detected failed login attempts and alerts. Include screenshots or copy-paste the output. Describe how you simulated the failed logins (e.g., using `sudo su` with an incorrect password repeatedly).

- **Reflections on Automation:** Discuss the benefits of using scripts for incident response. For example:

    - **Speed:** Automation allows you to quickly gather information and detect threats, saving valuable time.
    - **Consistency:** Scripts ensure that tasks are performed consistently and accurately every time.
    - **Efficiency:** Automation frees up human analysts to focus on more complex tasks.
    - **Scalability:** Scripts can be easily deployed to multiple systems, making it easier to manage large environments.

Bonus Challenge

Create a **Detection script** to automatically block an IP address after multiple failed login attempts.

Appendix A:

Part 1: Preparation Phase – Asset Inventory Script (PowerShell)

PowerShell

```
# Get Active Users
Write-Host "Active Users:"
Get-LocalUser | Where-Object {$_.Enabled -eq $true} | Select-Object Name,
LastLogon, Enabled | Format-Table -AutoSize

# Installed Software and Versions
Write-Host "`nInstalled Software:"
Get-WmiObject -Class Win32_Product | Select-Object Name, Version | Format-
Table -AutoSize

# Check for Missing Security Patches (Windows Update)
Write-Host "`nMissing Security Patches (Windows Update):"
$MissingUpdates = Get-WUInstall -List | Where-Object {$_.IsInstalled -eq
$false -and $_.Title -like "*Security Update*"}
if ($MissingUpdates) {
    $MissingUpdates | Select-Object Title, KBID | Format-Table -AutoSize
} else {
    Write-Host "No missing security updates found."
}

# (Alternative for Patch Check - using WSUS or other patch management)
# This would require more specific commands depending on your system.
# Example (hypothetical - adapt as needed):
# Get-WSUSUpdate -Status "NotApproved" | Select-Object Title, Classification
```

Appendix B:

Part 2: Detection Phase – Suspicious Login Monitoring (Python)

Python

```
import re
import time
from collections import defaultdict

# Log file path (adjust to your system)
log_file = "auth.log"  # Example: /var/log/auth.log on Linux

# Failed login threshold
threshold = 5
time_window = 600  # 10 minutes in seconds

# Dictionary to store failed login attempts (IP: [timestamps])
failed_attempts = defaultdict(list)

def analyze_logs():
    while True:
        try:
            with open(log_file, "r") as f:
                for line in f:
                    # Regex to extract IP and login status (adapt to your log
format)
                    match =
re.search(r"(?P<ip>\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}).*(Failed|failure|inval
id) password", line, re.IGNORECASE)
                    if match:
                        ip = match.group("ip")
                        failed_attempts[ip].append(time.time())

                        # Check for threshold
                        now = time.time()
                        recent_attempts = [t for t in failed_attempts[ip] if
now - t < time_window]
                        failed_attempts[ip] = recent_attempts  # Keep only
recent attempts
                        if len(recent_attempts) >= threshold:
                            print(f"ALERT: Potential brute-force attack from
IP: {ip}")
                            # You could add code here to block the IP (Bonus
Challenge)
            time.sleep(60)  # Check every minute
        except FileNotFoundError:
            print(f"Error: Log file '{log_file}' not found.")
            break
        except Exception as e:
            print(f"An error occurred: {e}")
            break

if __name__ == "__main__":
    analyze_logs()
```

Appendix C:

Incident Response Automation Report (Goby)

1. Introduction

Provide a brief introduction explaining the purpose of the report, the tasks performed, and the importance of automation in incident response.

2. Scripts Used

2.1 Preparation Script

- **Script Type**: (PowerShell, Python, Bash, Batch)
- **Script Code**: (Include the full script used for the preparation phase)
- **Purpose**: Explain what the script does (e.g., gathers system information, checks for missing patches, lists active users, etc.)

2.2 Detection Script

- **Script Type**: (PowerShell, Python, Bash, Batch)
- **Script Code**: (Include the full script used for detecting failed login attempts)
- **Purpose**: Explain how the script detects unauthorized access attempts and alerts the user.

3. Findings

3.1 System Information Gathering

- **List of Active Users:** (Output or screenshot from the preparation script)
- **List of Installed Software and Versions:** (Output or screenshot from the preparation script)
- **Missing Security Patches:** (Details from the preparation script)

3.2 Failed Login Attempts Detection

- **Output from Detection Script:** (Copy-paste or screenshot showing failed login attempts)
- **Simulation of Failed Logins:** Describe how failed logins were simulated (e.g., entering incorrect passwords multiple times).
- **Analysis of Results:** Discuss any patterns found, such as repeated failed attempts from a specific IP address or account.

4. Reflections on Automation in Incident Response

4.1 Benefits of Automation

- **Speed:** Explain how scripts accelerate detection and response.
- **Consistency:** Discuss how automation ensures accurate and repeatable results.
- **Efficiency:** Highlight how automation reduces manual workload.
- **Scalability:** Explain how scripts can be deployed across multiple systems for large-scale monitoring.

4.2 Lessons Learned

- Challenges faced while writing and running the scripts.
- Improvements or modifications that could enhance the scripts.
- Insights gained about security monitoring and incident response.

5. Conclusion

Summarize key findings and reaffirm the importance of automation in enhancing security operations.

6. References (If Applicable)

Cite any resources, documentation, or tutorials used to develop the scripts.

IP Blocking (Extra Effort)

You can add IP blocking to the Python script. The method depends on your operating system and network setup. Here's a basic example using `iptables` (Linux):

Python

```
# ... (previous code) ...
if len(recent_attempts) >= threshold:
    print(f"ALERT: Potential brute-force attack from IP: {ip}")
    # Block the IP (iptables example - requires root privileges)
    import subprocess
    try:
        subprocess.run(["iptables", "-A", "INPUT", "-s", ip, "-j", "DROP"],
check=True)  # Adapt to your needs
        print(f"IP {ip} blocked.")
    except subprocess.CalledProcessError as e:
        print(f"Error blocking IP: {e}")

# ... (rest of the code) ...
```

Important Notes:

- **Log File Format:** The Python script's regex for parsing logs *must* match the format of your system's log files. Examine your logs carefully to determine the correct pattern.
- **Permissions:** Running the IP blocking part of the script will require root or administrator privileges.
- **Testing:** Test your scripts thoroughly on a test machine before using them in a production environment.
- **Security:** Be cautious about how you implement IP blocking. Make sure you have a way to unblock legitimate users if necessary. Consider using more sophisticated intrusion prevention systems (IPS) for production environments.
- **Adaptation:** The scripts provided are examples. You may need to modify them to fit your specific requirements and environment. For example, you might use different tools for patch management or different log formats.