Incident Response Lab Report: Python Automation for Brute-Force Detection and Alerting

By Marcus Dunn

Executive Summary

Lab notes for deploying and running a SOC automation for alerting on brute force login attempts with Slack notification. In this activity, adversarial behavior was simulated with persistent SSH login attempts that failed repeatedly. This demonstrated how automation and custom scripts bridged a gap between raw log data and real-time human-readable alerting. This activity reinforced hands-on expertise with scripting, log parsing, environment control, and alerting with security.

Lab Overview and Objectives

- Create a Python program that searches for brute-force login attempts in Linux auth logs.
- Develop and facilitate a virtual space to hold the project
- Communicate with Slack Webhooks securely using requests and python-dotenv
- Deal with environment and script errors when executing
- Route alert messages formatted for a Slack intake channel upon detection of malicious activity
- Mimic brute-force attack patterns to check for detection accuracy
- Troubleshoot permission, syntax, and logic errors during development

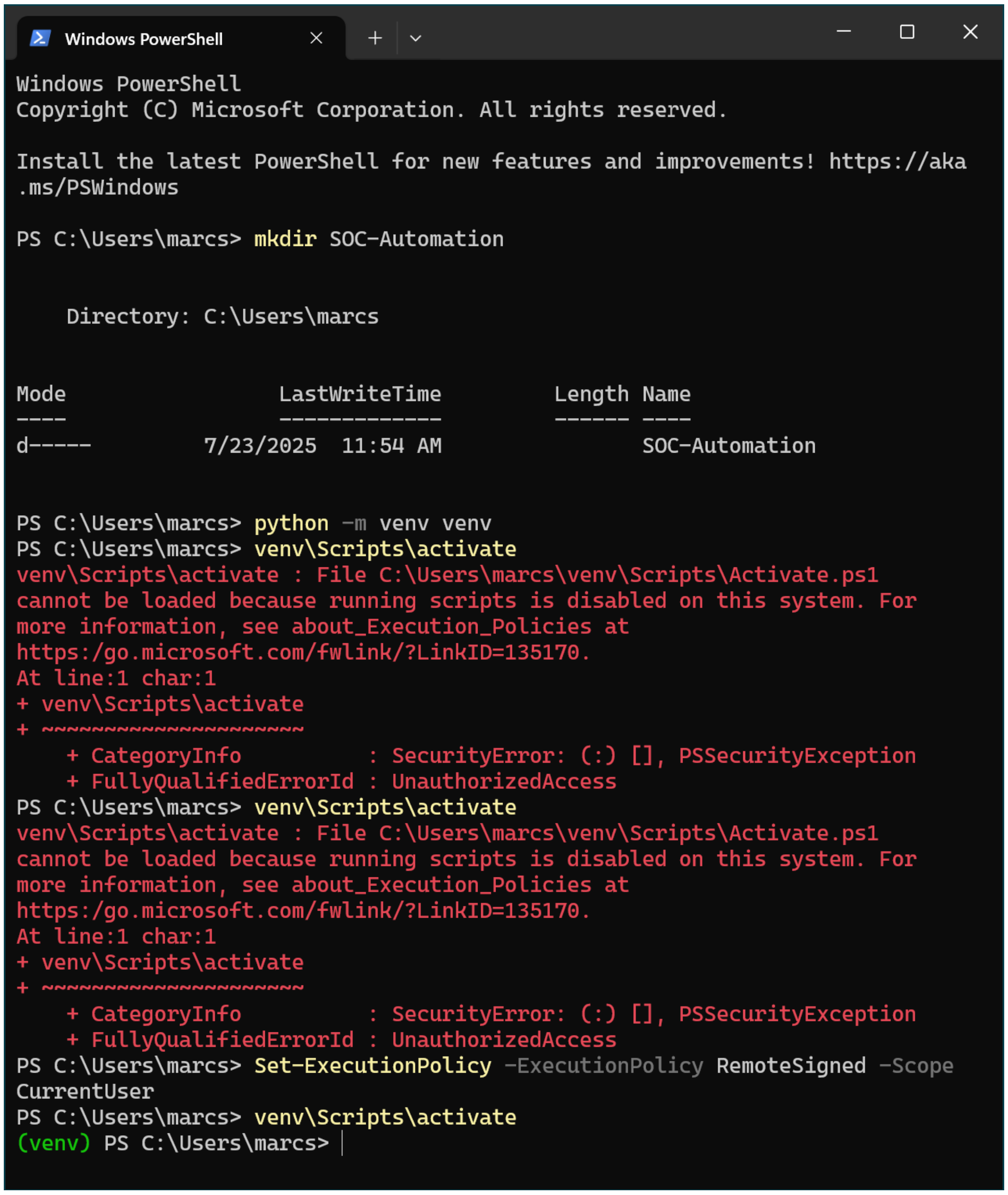
Environment Architecture

- Host Machine: Windows 11
- Tools & Tech: Python 3.12, PowerShell, Visual Studio Code
- Dependencies: requests, python-dotenv
- Notification Platform: Slack (SOC Automation Tool App)
- Input Source: Linux-style auth.log brute-force logs (manually transferred)
- Output Channel: Slack (#soc-automation-intake)

Lab Setup & Configuration

To isolate dependencies and control Python versioning, the project was set up in a dedicated virtual environment using PowerShell.

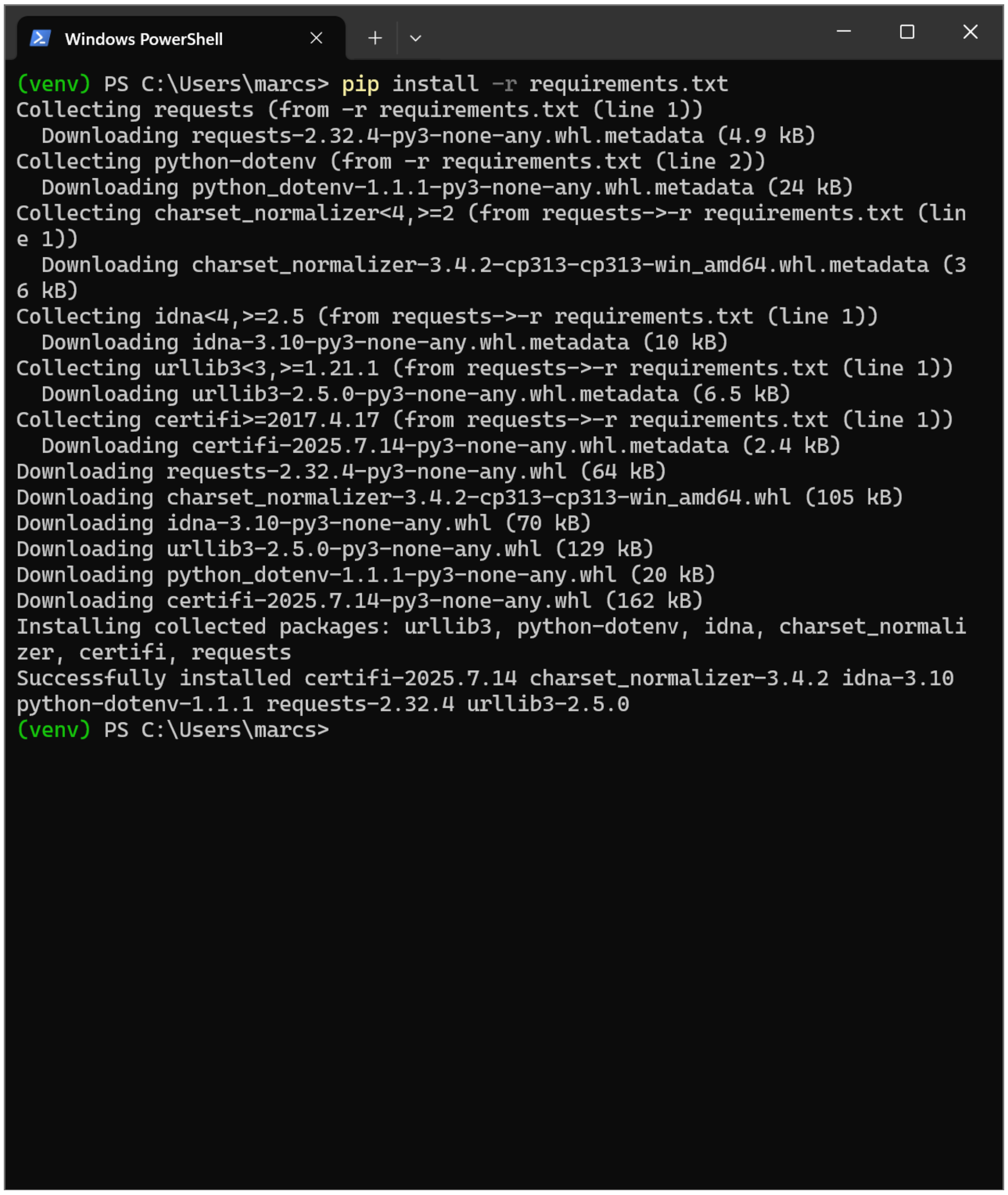
Virtual Environment Setup (with Execution Policy Troubleshooting):



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka
.ms/PSWindows

PS C:\Users\marcs> mkdir SOC-Automation


    Directory: C:\Users\marcs


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         7/23/2025  11:54 AM                SOC-Automation


PS C:\Users\marcs> python -m venv venv
PS C:\Users\marcs> venv\Scripts\activate
venv\Scripts\activate : File C:\Users\marcs\venv\Scripts\Activate.ps1
cannot be loaded because running scripts is disabled on this system. For
more information, see about_Execution_Policies at
https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ venv\Scripts\activate
+ ~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\marcs> venv\Scripts\activate
venv\Scripts\activate : File C:\Users\marcs\venv\Scripts\Activate.ps1
cannot be loaded because running scripts is disabled on this system. For
more information, see about_Execution_Policies at
https:/go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ venv\Scripts\activate
+ ~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : SecurityError: (:) [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\marcs> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope
CurrentUser
PS C:\Users\marcs> venv\Scripts\activate
(venv) PS C:\Users\marcs>
```

requirements.txt & Dependency Install:

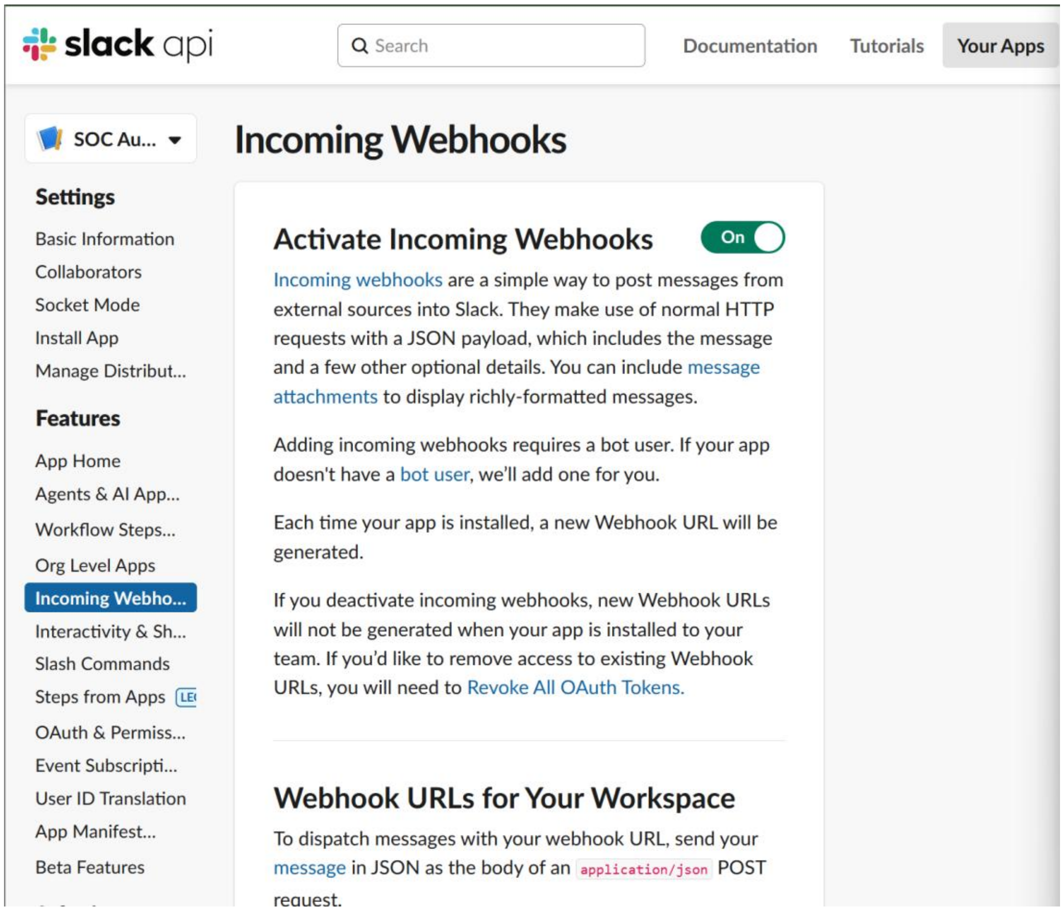
```
requests
python-dotenv
```



```
(venv) PS C:\Users\marcs> pip install -r requirements.txt
Collecting requests (from -r requirements.txt (line 1))
  Downloading requests-2.32.4-py3-none-any.whl.metadata (4.9 kB)
Collecting python-dotenv (from -r requirements.txt (line 2))
  Downloading python_dotenv-1.1.1-py3-none-any.whl.metadata (24 kB)
Collecting charset_normalizer<4,>=2 (from requests->-r requirements.txt (lin
e 1))
  Downloading charset_normalizer-3.4.2-cp313-cp313-win_amd64.whl.metadata (3
6 kB)
Collecting idna<4,>=2.5 (from requests->-r requirements.txt (line 1))
  Downloading idna-3.10-py3-none-any.whl.metadata (10 kB)
Collecting urllib3<3,>=1.21.1 (from requests->-r requirements.txt (line 1))
  Downloading urllib3-2.5.0-py3-none-any.whl.metadata (6.5 kB)
Collecting certifi>=2017.4.17 (from requests->-r requirements.txt (line 1))
  Downloading certifi-2025.7.14-py3-none-any.whl.metadata (2.4 kB)
Downloading requests-2.32.4-py3-none-any.whl (64 kB)
Downloading charset_normalizer-3.4.2-cp313-cp313-win_amd64.whl (105 kB)
Downloading idna-3.10-py3-none-any.whl (70 kB)
Downloading urllib3-2.5.0-py3-none-any.whl (129 kB)
Downloading python_dotenv-1.1.1-py3-none-any.whl (20 kB)
Downloading certifi-2025.7.14-py3-none-any.whl (162 kB)
Installing collected packages: urllib3, python-dotenv, idna, charset_normali
zer, certifi, requests
Successfully installed certifi-2025.7.14 charset_normalizer-3.4.2 idna-3.10
python-dotenv-1.1.1 requests-2.32.4 urllib3-2.5.0
(venv) PS C:\Users\marcs>
```

Slack integration was handled via the Slack API using Incoming Webhooks, which required setting up an app with permissions and generating a unique webhook URL stored securely in a .env file.

Slack App Configuration:



Incoming Webhooks

Activate Incoming Webhooks

Incoming webhooks are a simple way to post messages from external sources into Slack. They make use of normal HTTP requests with a JSON payload, which includes the message and a few other optional details. You can include message attachments to display richly-formatted messages.

Adding incoming webhooks requires a bot user. If your app doesn't have a bot user, we'll add one for you.

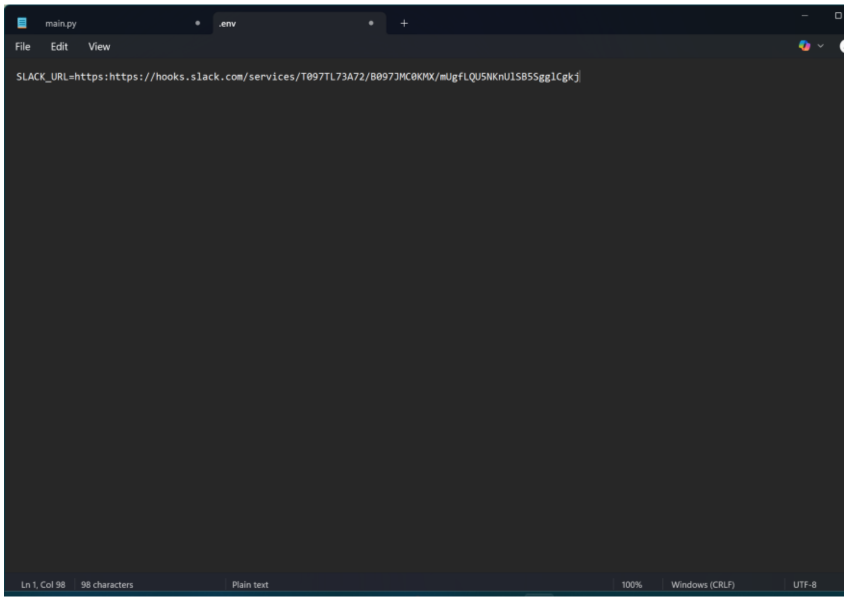
Each time your app is installed, a new Webhook URL will be generated.

If you deactivate incoming webhooks, new Webhook URLs will not be generated when your app is installed to your team. If you'd like to remove access to existing Webhook URLs, you will need to Revoke All OAuth Tokens.

Webhook URLs for Your Workspace

To dispatch messages with your webhook URL, send your message in JSON as the body of an `application/json` POST request.

Webhook Credential in .env

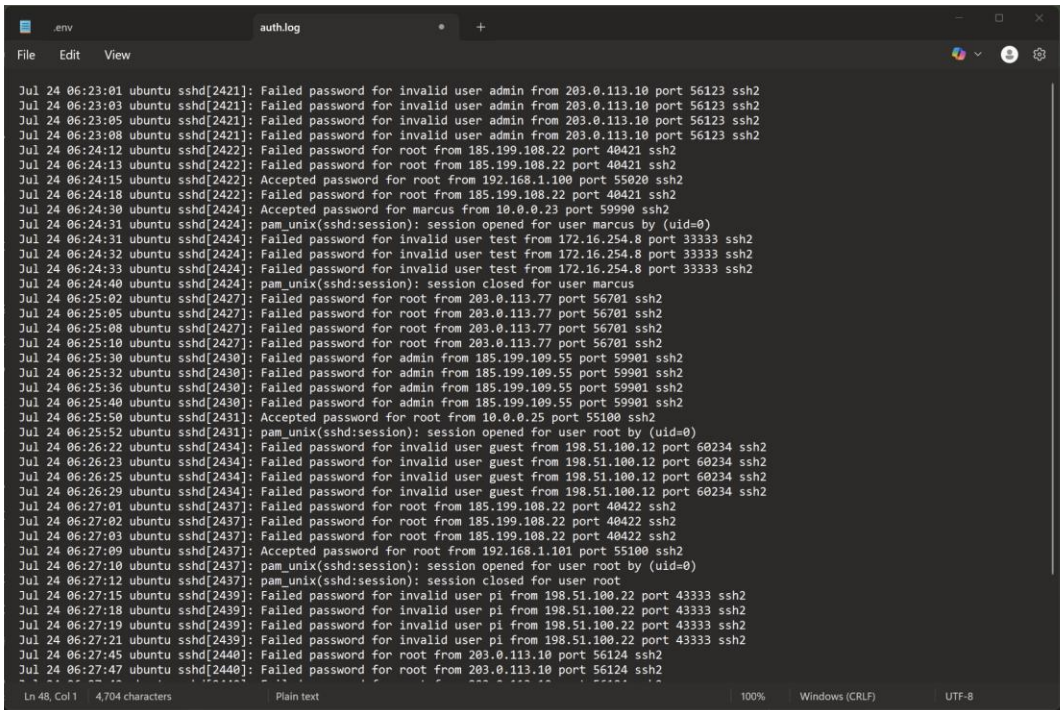


```
SLACK_URL=https:https://hooks.slack.com/services/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
```

Brute-Force Simulation & Detection

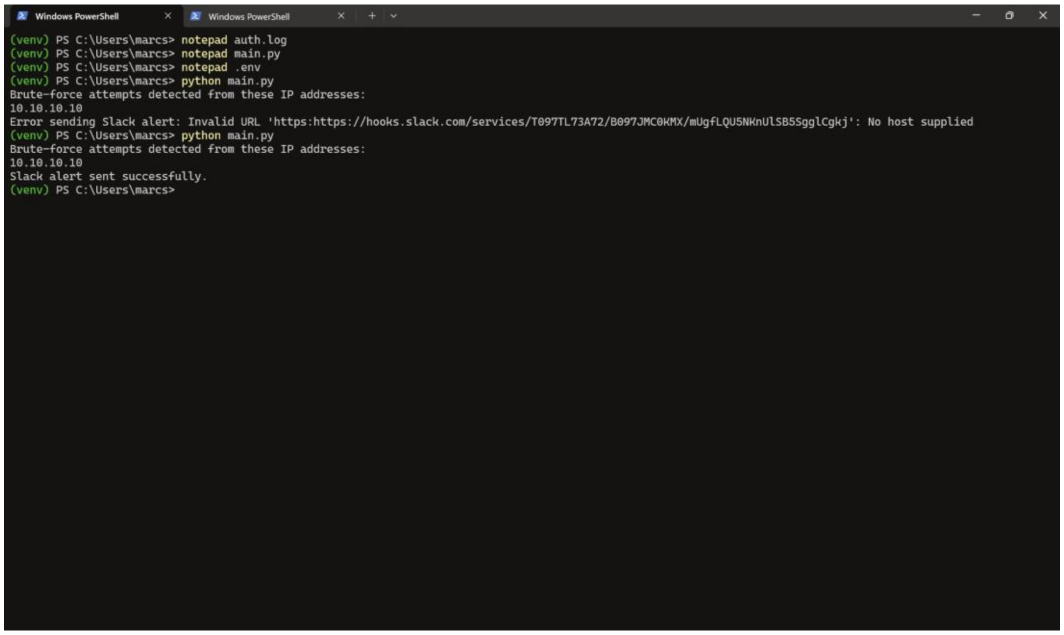
Attack traffic was simulated using brute-force type login attempts in auth.log that mimicked failed SSH logins from different IP addresses. Pcap was parsed with a parser that extracted distinct offenders from patterns like:
Failed password for root from 203.0.113.10 port 56123 ssh2

Sample Log File Opened in Notepad:



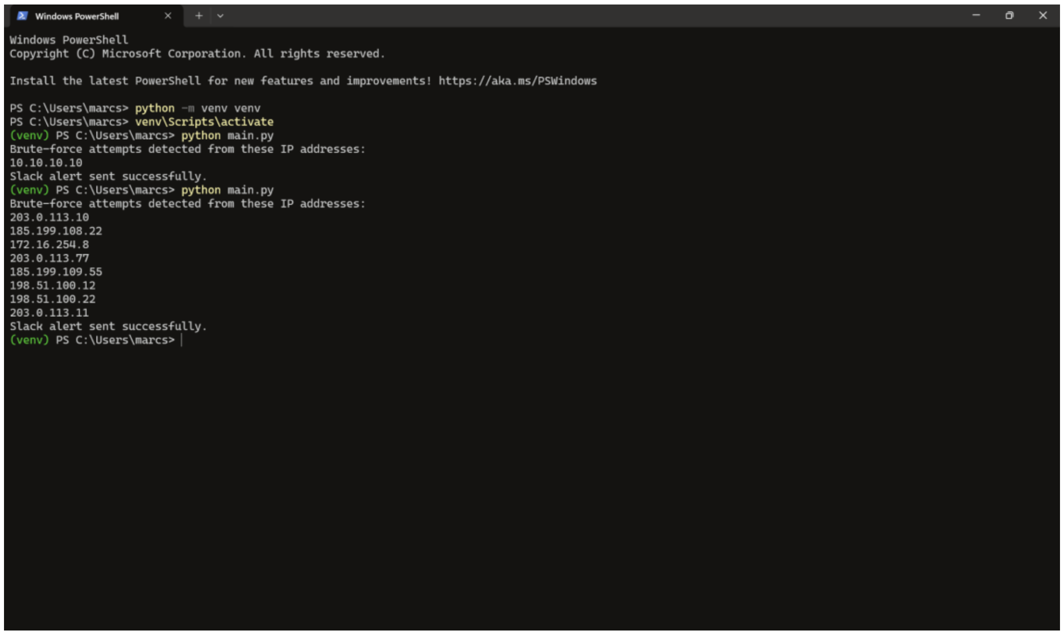
```
Jul 24 06:23:01 ubuntu sshd[2421]: Failed password for invalid user admin from 203.0.113.10 port 56123 ssh2
Jul 24 06:23:03 ubuntu sshd[2421]: Failed password for invalid user admin from 203.0.113.10 port 56123 ssh2
Jul 24 06:23:05 ubuntu sshd[2421]: Failed password for invalid user admin from 203.0.113.10 port 56123 ssh2
Jul 24 06:23:08 ubuntu sshd[2421]: Failed password for invalid user admin from 203.0.113.10 port 56123 ssh2
Jul 24 06:24:12 ubuntu sshd[2422]: Failed password for root from 185.199.108.22 port 40421 ssh2
Jul 24 06:24:13 ubuntu sshd[2422]: Failed password for root from 185.199.108.22 port 40421 ssh2
Jul 24 06:24:15 ubuntu sshd[2422]: Accepted password for root from 192.168.1.100 port 55020 ssh2
Jul 24 06:24:18 ubuntu sshd[2422]: Failed password for root from 185.199.108.22 port 40421 ssh2
Jul 24 06:24:30 ubuntu sshd[2424]: Accepted password for marcus from 10.0.0.23 port 59990 ssh2
Jul 24 06:24:31 ubuntu sshd[2424]: pam_unix(sshd:session): session opened for user marcus by (uid=0)
Jul 24 06:24:31 ubuntu sshd[2424]: Failed password for invalid user test from 172.16.254.8 port 33333 ssh2
Jul 24 06:24:32 ubuntu sshd[2424]: Failed password for invalid user test from 172.16.254.8 port 33333 ssh2
Jul 24 06:24:33 ubuntu sshd[2424]: Failed password for invalid user test from 172.16.254.8 port 33333 ssh2
Jul 24 06:24:40 ubuntu sshd[2424]: pam_unix(sshd:session): session closed for user marcus
Jul 24 06:25:02 ubuntu sshd[2427]: Failed password for root from 203.0.113.77 port 56701 ssh2
Jul 24 06:25:05 ubuntu sshd[2427]: Failed password for root from 203.0.113.77 port 56701 ssh2
Jul 24 06:25:08 ubuntu sshd[2427]: Failed password for root from 203.0.113.77 port 56701 ssh2
Jul 24 06:25:10 ubuntu sshd[2427]: Failed password for root from 203.0.113.77 port 56701 ssh2
Jul 24 06:25:30 ubuntu sshd[2430]: Failed password for admin from 185.199.109.55 port 59901 ssh2
Jul 24 06:25:32 ubuntu sshd[2430]: Failed password for admin from 185.199.109.55 port 59901 ssh2
Jul 24 06:25:36 ubuntu sshd[2430]: Failed password for admin from 185.199.109.55 port 59901 ssh2
Jul 24 06:25:40 ubuntu sshd[2430]: Failed password for admin from 185.199.109.55 port 59901 ssh2
Jul 24 06:25:50 ubuntu sshd[2431]: Accepted password for root from 10.0.0.25 port 55100 ssh2
Jul 24 06:25:52 ubuntu sshd[2431]: pam_unix(sshd:session): session opened for user root by (uid=0)
Jul 24 06:26:22 ubuntu sshd[2434]: Failed password for invalid user guest from 198.51.100.12 port 60234 ssh2
Jul 24 06:26:23 ubuntu sshd[2434]: Failed password for invalid user guest from 198.51.100.12 port 60234 ssh2
Jul 24 06:26:25 ubuntu sshd[2434]: Failed password for invalid user guest from 198.51.100.12 port 60234 ssh2
Jul 24 06:26:29 ubuntu sshd[2434]: Failed password for invalid user guest from 198.51.100.12 port 60234 ssh2
Jul 24 06:27:01 ubuntu sshd[2437]: Failed password for root from 185.199.108.22 port 40422 ssh2
Jul 24 06:27:02 ubuntu sshd[2437]: Failed password for root from 185.199.108.22 port 40422 ssh2
Jul 24 06:27:03 ubuntu sshd[2437]: Failed password for root from 185.199.108.22 port 40422 ssh2
Jul 24 06:27:09 ubuntu sshd[2437]: Accepted password for root from 192.168.1.101 port 55100 ssh2
Jul 24 06:27:10 ubuntu sshd[2437]: pam_unix(sshd:session): session opened for user root by (uid=0)
Jul 24 06:27:12 ubuntu sshd[2437]: pam_unix(sshd:session): session closed for user root
Jul 24 06:27:15 ubuntu sshd[2439]: Failed password for invalid user pi from 198.51.100.22 port 43333 ssh2
Jul 24 06:27:18 ubuntu sshd[2439]: Failed password for invalid user pi from 198.51.100.22 port 43333 ssh2
Jul 24 06:27:19 ubuntu sshd[2439]: Failed password for invalid user pi from 198.51.100.22 port 43333 ssh2
Jul 24 06:27:21 ubuntu sshd[2439]: Failed password for invalid user pi from 198.51.100.22 port 43333 ssh2
Jul 24 06:27:45 ubuntu sshd[2440]: Failed password for root from 203.0.113.10 port 56124 ssh2
Jul 24 06:27:47 ubuntu sshd[2440]: Failed password for root from 203.0.113.10 port 56124 ssh2
```

Script Execution and Output:



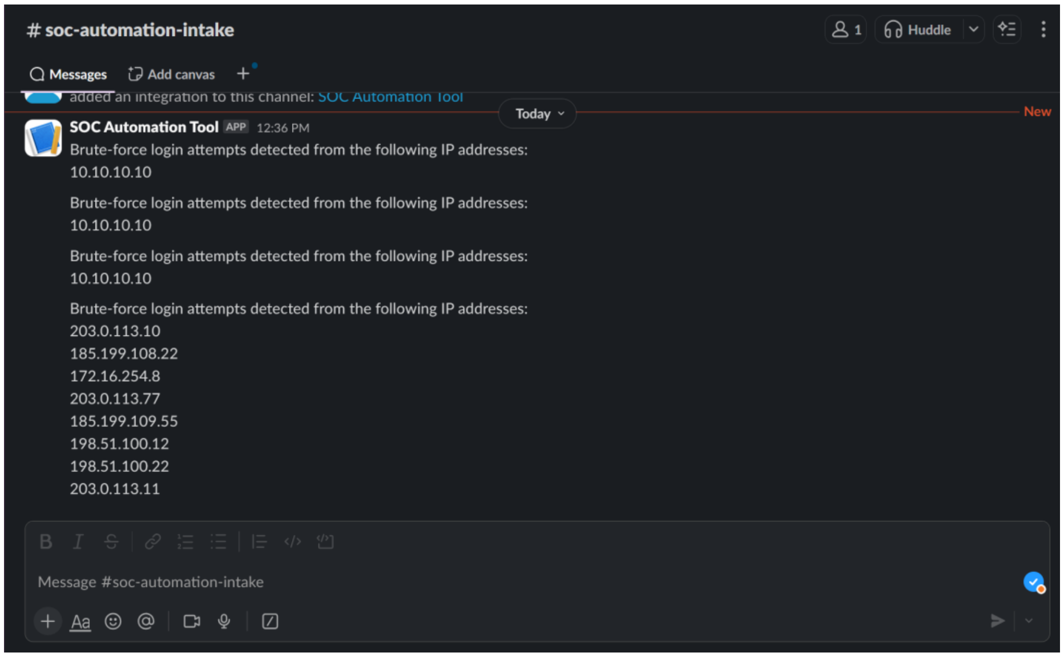
```
(venv) PS C:\Users\marcs> notepad auth.log
(venv) PS C:\Users\marcs> notepad main.py
(venv) PS C:\Users\marcs> notepad .env
(venv) PS C:\Users\marcs> python main.py
Brute-force attempts detected from these IP addresses:
10.10.10.10
Error sending Slack alert: Invalid URL 'https:https://hooks.slack.com/services/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': No host supplied
(venv) PS C:\Users\marcs> python main.py
Brute-force attempts detected from these IP addresses:
10.10.10.10
Slack alert sent successfully.
(venv) PS C:\Users\marcs>
```

Console Output - Detection Working:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\marcs> python -m venv venv
PS C:\Users\marcs> venv\Scripts\activate
(venv) PS C:\Users\marcs> python main.py
Brute-force attempts detected from these IP addresses:
10.10.10.10
Slack alert sent successfully.
(venv) PS C:\Users\marcs> python main.py
Brute-force attempts detected from these IP addresses:
203.0.113.10
185.199.108.22
172.16.254.8
203.0.113.77
185.199.109.55
198.51.100.12
198.51.100.22
203.0.113.11
Slack alert sent successfully.
(venv) PS C:\Users\marcs>
```

Final Slack Notification in Target Channel:



# soc-automation-intake
1
Huddle
Messages
Add canvas
added an integration to this channel: SOC Automation Tool
Today
New
SOC Automation Tool APP 12:36 PM
Message #soc-automation-intake

```
Brute-force login attempts detected from the following IP addresses:
10.10.10.10

Brute-force login attempts detected from the following IP addresses:
10.10.10.10

Brute-force login attempts detected from the following IP addresses:
10.10.10.10

Brute-force login attempts detected from the following IP addresses:
203.0.113.10
185.199.108.22
172.16.254.8
203.0.113.77
185.199.109.55
198.51.100.12
198.51.100.22
203.0.113.11
```

Troubleshooting & Problem Solving

- PowerShell script execution error: Resolved by adjusting execution policy to RemoteSigned
- Malformed webhook URL: Corrected after reformatting .env and reissuing a valid Slack URL
- Silent failures in Python output: Fixed by explicitly printing debug output and verifying

auth.log path
- Duplicate alerts in Slack: Tweaked logic to send one alert per unique batch of IPs

Lessons Learned

- Hands-on practice with Python scripting for security automation
- Set up a functioning Slack alerting pipeline with environment-secured webhooks
- Hands-on log parsing and brute-force pattern recognition on sample data
- Enhanced troubleshooting skill encompassing both PowerShell and Python environment quirks
- Created a working proof-of-concept SOC intake instrument that eliminates real-time manual review gaps

Conclusion

This project reinforced the value of custom automation in the SOC workflow. From parsing low-level logs to triggering human-readable alerts, each step deepened my understanding of cybersecurity operations and incident response tooling. The simplicity of the tool doesn't undermine its power — it reflects a modular, scalable approach to threat detection and alerting.

Python Script: Brute-Force Detection & Slack Alerting

```
import re
from collections import defaultdict
import os
import requests
from dotenv import load_dotenv

# Load sensitive configuration from environment variables.
load_dotenv()

def parse_log(filename):
    pattern = re.compile(
        r'(?P<month>\w{3}) (?P<day>\d{1,2}) (?P<time>\d{2}:\d{2}:\d{2})
.*sshd.*Failed password for .* from (?P<ip>\d+\.\d+\.\d+\.\d+)'
    )
    failed_logins = []
    try:
        with open(filename, 'r') as file:
            for line in file:
                match = pattern.search(line)
                if match:
                    failed_logins.append({
                        'ip': match.group('ip'),
                        'datetime': f"{match.group('month')}
{match.group('day')} {match.group('time')}"
                    })
    except FileNotFoundError:
        print(f"Log file not found: {filename}")
    except Exception as e:
        print(f"Error reading log file: {e}")
    return failed_logins
```

```
def detect_bruteforce(logins, threshold=3):
    ip_counts = defaultdict(int)
    for entry in logins:
        ip_counts[entry['ip']] += 1
    return [ip for ip, count in ip_counts.items() if count >= threshold]

def send_slack_alert(attackers):
    webhook_url = os.getenv("SLACK_URL")
    if not webhook_url:
        print("Slack webhook URL not set. Set SLACK_URL in your environment.")
        return
    message = "Brute-force login attempts detected from the following IP 
addresses:\n" + "\n".join(attackers)
    data = {"text": message}
    try:
        resp = requests.post(webhook_url, json=data)
        if resp.status_code == 200:
            print("Slack alert sent successfully.")
        else:
            print(f"Failed to send Slack alert (HTTP {resp.status_code}): 
{resp.text}")
    except Exception as e:
        print(f"Error sending Slack alert: {e}")

if __name__ == '__main__':
    logins = parse_log('auth.log')
    attackers = detect_bruteforce(logins, threshold=3)
    if attackers:
        print("Brute-force attempts detected from these IP addresses:")
        for ip in attackers:
            print(ip)
        send_slack_alert(attackers)
    else:
        print("No brute-force attacks detected in the provided log.")
```