

# Python Script to Analyze Security Logs Using Natural Language Processing (NLP)

**1. Create a file** and open it in the Nano editor using the command: -

```
$ nano NLP_Security_Logs.py
```

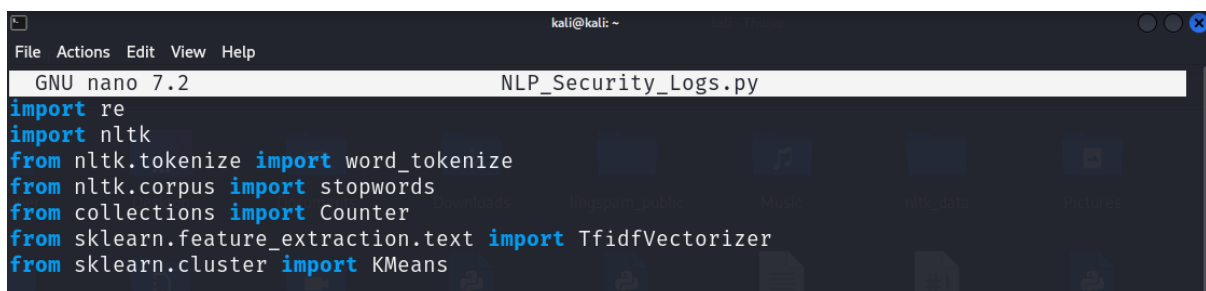


```
(kali@kali)-[~]  
$ nano NLP_Security_Logs.py
```

This will open the nano editor with the mentioned file name and type. This is where you can write down the code. You can obtain this code from

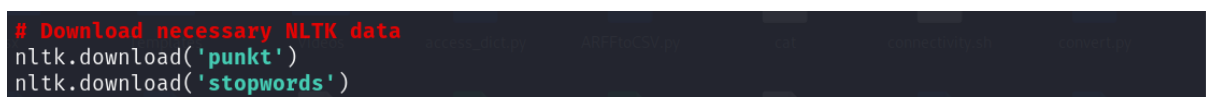
[https://github.com/avnishnaithani/pythonforsecurity/blob/main/Domain%202/NLP\\_Security\\_Logs.py](https://github.com/avnishnaithani/pythonforsecurity/blob/main/Domain%202/NLP_Security_Logs.py)

**2. Importing Required Libraries:** These lines in the screenshot below import the necessary tools. **re** is for regular expressions, **nlTK** is a natural language toolkit, **Counter** helps count things easily, and **sklearn** provides machine learning tools.



```
File Actions Edit View Help  
GNU nano 7.2 NLP_Security_Logs.py  
import re  
import nltk  
from nltk.tokenize import word_tokenize  
from nltk.corpus import stopwords  
from collections import Counter  
from sklearn.feature_extraction.text import TfidfVectorizer  
from sklearn.cluster import KMeans
```

**3. Downloading NLTK Data:** This downloads necessary data for NLTK to work properly.



```
# Download necessary NLTK data  
nltk.download('punkt')  
nltk.download('stopwords')
```

**4. Sample Data:** This is our pretend security log data. In a real scenario, you'd read this from files.

```
# Sample security log entries
security_logs = [
    "2023-10-01 08:30:15 Failed login attempt from IP 192.168.1.100",
    "2023-10-01 09:15:22 Successful login by user admin",
    "2023-10-01 10:45:30 Firewall blocked connection from IP 203.0.113.42",
    "2023-10-01 11:20:18 File deletion in /sensitive/data by user john",
    "2023-10-01 12:05:45 Multiple failed login attempts from IP 192.168.1.100",
    "2023-10-01 13:10:33 Unusual outbound traffic detected to IP 198.51.100.77",
    "2023-10-01 14:30:27 System update installed successfully",
    "2023-10-01 15:45:12 New user account created: alice",
    "2023-10-01 16:20:55 Suspicious file quarantined: malware.exe",
    "2023-10-01 17:05:40 Failed login attempt from IP 203.0.113.42"
]
```

**5. Extracting IP Addresses:** This function uses a regular expression to find all IP addresses in the logs.

```
def extract_ip_addresses(logs):
    ip_pattern = r'\b(?:\d{1,3}\.){3}\d{1,3}\b'
    ip_addresses = [ip for log in logs for ip in re.findall(ip_pattern, log)]
    return ip_addresses
```

**6. Extracting Usernames:** Similar to IP extraction, this finds all usernames mentioned after the word "user".

```
def extract_usernames(logs):
    username_pattern = r'user (\w+)'
    usernames = [match.group(1) for log in logs for match in re.finditer(username_pattern, log)]
    return usernames
```

**7. Preprocessing Logs:** This function cleans up the log text by: -

- Converting data to lowercase.
- Splitting data into words (tokenization).
- Removing punctuation and common words (stop words).
- Joining the remaining words back into a string.

The functions used for this purpose are present in the nltk library, which is most crucial for implementing Natural Language Processing.

```
def preprocess_logs(logs):
    stop_words = set(stopwords.words('english'))
    processed_logs = []
    for log in logs:
        tokens = word_tokenize(log.lower())
        tokens = [token for token in tokens if token.isalnum() and token not in stop_words]
        processed_logs.append(' '.join(tokens))
    return processed_logs
```

**8. Identifying Common Events:** This counts all words and returns the most common ones.

```
def identify_common_events(logs, top_n=5):
    words = [word for log in logs for word in log.split()]
    return Counter(words).most_common(top_n)
```

**9. Clustering Logs:** This is more advanced. It:

- Converts logs to a numerical format (TF-IDF).
- Uses K-means (Machine Learning Algorithm) to group similar logs together.

```
def cluster_logs(logs, n_clusters=3):
    vectorizer = TfidfVectorizer()
    X = vectorizer.fit_transform(logs)
    kmeans = KMeans(n_clusters=n_clusters)
    kmeans.fit(X)
    return kmeans.labels_
```

**10. Main Function:** This function runs all our analysis steps and prints the results.

```
def main():
    print("Extracting IP addresses:")
    ip_addresses = extract_ip_addresses(security_logs)
    print(ip_addresses)

    print("\nExtracting usernames:")
    usernames = extract_usernames(security_logs)
    print(usernames)

    processed_logs = preprocess_logs(security_logs)

    print("\nMost common events:")
    common_events = identify_common_events(processed_logs)
    for event, count in common_events:
        print(f"{event}: {count}")

    print("\nClustering logs:")
    clusters = cluster_logs(processed_logs)
    for i, (log, cluster) in enumerate(zip(security_logs, clusters)):
        print(f"Log {i + 1} - Cluster {cluster}: {log}")

    print("\nPotential security insights:")
    ip_counter = Counter(ip_addresses)
    for ip, count in ip_counter.items():
        if count > 1:
            print(f"Multiple events from IP {ip}")

    if 'failed login attempt' in ' '.join(security_logs).lower():
        print("Failed login attempts detected")

    if 'suspicious' in ' '.join(security_logs).lower():
        print("Suspicious activity detected")

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

**11. Convert the file to an executable one using the command: -**

**\$ chmod +x NLP\_Security\_Logs.py**

```
(kali㉿kali)-[~]
$ chmod +x NLP_Security_Logs.py
```

**12. Finally execute the program. Use the command mentioned below for this: -**

**\$ python3 NLP\_Security\_Logs.py**

The output obtained is shown below: -

```
(kali㉿kali)-[~]
$ python NLP_Security_Logs.py
[nltk_data] Downloading package punkt to /home/kali/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /home/kali/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
Extracting IP addresses:
['192.168.1.100', '203.0.113.42', '192.168.1.100', '198.51.100.77', '203.0.113.42']

Extracting usernames:
['admin', 'john', 'account']

Most common events:
ip: 5
login: 4
failed: 3
user: 3
attempt: 2

Clustering logs:
Log 1 - Cluster 2: 2023-10-01 08:30:15 Failed login attempt from IP 192.168.1.100
Log 2 - Cluster 1: 2023-10-01 09:15:22 Successful login by user admin
Log 3 - Cluster 2: 2023-10-01 10:45:30 Firewall blocked connection from IP 203.0.113.42
Log 4 - Cluster 1: 2023-10-01 11:20:18 File deletion in /sensitive/data by user john
Log 5 - Cluster 2: 2023-10-01 12:05:45 Multiple failed login attempts from IP 192.168.1.100
Log 6 - Cluster 0: 2023-10-01 13:10:33 Unusual outbound traffic detected to IP 198.51.100.7
Log 7 - Cluster 0: 2023-10-01 14:30:27 System update installed successfully
Log 8 - Cluster 1: 2023-10-01 15:45:12 New user account created: alice
Log 9 - Cluster 1: 2023-10-01 16:20:55 Suspicious file quarantined: malware.exe
Log 10 - Cluster 2: 2023-10-01 17:05:40 Failed login attempt from IP 203.0.113.42

Potential security insights:
Multiple events from IP 192.168.1.100
Multiple events from IP 203.0.113.42
Failed login attempts detected
Suspicious activity detected
```

Based on the logs we have detected failed login attempts and hence this NLP powered system has detected suspicious activities from some users. These users can be tracked by their IP addresses captured by this model.

So, we have seen that our model is working on the log data entered in the code. This means our script (powered by Natural Language Processing) is doing the job of reading the logs for us. **Therefore, we have automated the process of reading logs.**

In real world applications SIEM (Security Information and Event Management) tools collect log data from multiple sources and aggregates it across the IT infrastructure into a centralized platform where security analysts can review it. We have used AI (Specifically

NLP) such that we can just load the collected logs into the model and it will do the job of finding issues.

This way AI complements human analysts by picking up on some manual errors, automating the process and giving more time to security professionals to focus on other important tasks.