02_Clustering_for_Attack_Pattern_Discovery

April 6, 2025

1 Clustering for Attack Pattern Discovery

This notebook demonstrates how to use clustering techniques to discover attack patterns in security logs. We'll analyze SecurityOnion logs to identify groups of similar events that might indicate malicious activity.

[12]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans, DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import duckdb

# Set plot style
plt.style.use('ggplot')
sns.set(style="whitegrid")

# Connect to the DuckDB database
conn = duckdb.connect('../db/security_logs.duckdb')
```

1.1 1. Data Loading and Preparation

[13]:
```
# Load data from DuckDB - use the correct table schema
query = """
SELECT * FROM security_logs
WHERE protocol IS NOT NULL
LIMIT 100000
"""

df = conn.execute(query).fetchdf()
df.head()
```

[13]:
```
   event_id               timestamp      source_ip  source_port  \
0         1 2025-03-28 00:59:59.649  35.203.210.32        50930
1         2 2025-03-28 00:59:59.649  35.203.210.32        50930
```

```
2         3 2025-03-28 00:59:58.402  38.132.109.168        57070
3         4 2025-03-28 00:59:58.402  38.132.109.168        57070
4         5 2025-03-28 00:59:56.032    213.32.32.91        33270

          dest_ip  dest_port             action protocol  bytes  \
0  195.201.244.27      47466  chs add blocker 3      tcp     44
1  195.201.244.27      47466  chs dropped input      tcp     44
2  188.40.207.210       1723  chs dropped input      tcp     40
3  188.40.207.210       1723  chs add blocker 3      tcp     40
4  188.40.207.214       3400  chs add blocker 3      tcp     60

          country   log_date                      source_file raw_data
0  United Kingdom 2025-03-28  data_log_firewall_2025-03-28.json     None
1  United Kingdom 2025-03-28  data_log_firewall_2025-03-28.json     None
2   United States 2025-03-28  data_log_firewall_2025-03-28.json     None
3   United States 2025-03-28  data_log_firewall_2025-03-28.json     None
4          France 2025-03-28  data_log_firewall_2025-03-28.json     None
```

[14]:

```python
# Check data information
print(f"Shape: {df.shape}")
print("\nData types:")
print(df.dtypes)
print("\nMissing values:")
print(df.isnull().sum())
```

```
Shape: (100000, 13)

Data types:
event_id                int32
timestamp      datetime64[us]
source_ip              object
source_port             Int32
dest_ip                object
dest_port               Int32
action                 object
protocol               object
bytes                   int32
country                object
log_date       datetime64[us]
source_file            object
raw_data               object
dtype: object

Missing values:
event_id           0
timestamp          0
source_ip          0
source_port     1071
```

```
dest_ip              0
dest_port         1071
action               0
protocol             0
bytes                0
country             33
log_date             0
source_file          0
raw_data        100000
dtype: int64
```

1.2 2. Feature Engineering

Let's create features that will be useful for clustering security events.

[15]:
```python
# Create more feature engineering functions
def extract_features(df):
    features_df = df.copy()

    # Convert timestamps to datetime if needed
    if 'timestamp' in features_df.columns and not pd.api.types.
 ↪is_datetime64_any_dtype(features_df['timestamp']):
        features_df['timestamp'] = pd.to_datetime(features_df['timestamp'])

    # Extract time-based features
    if 'timestamp' in features_df.columns:
        features_df['hour_of_day'] = features_df['timestamp'].dt.hour
        features_df['day_of_week'] = features_df['timestamp'].dt.dayofweek
        features_df['is_weekend'] = features_df['day_of_week'].apply(lambda x:␣
 ↪1 if x >= 5 else 0)
        features_df['is_business_hours'] = features_df['hour_of_day'].
 ↪apply(lambda x: 1 if 9 <= x <= 17 else 0)

    # IP address features (if available)
    if 'source_ip' in features_df.columns:
        # Convert IP to numeric representation (simplified)
        features_df['source_ip_is_private'] = features_df['source_ip'].apply(
            lambda x: 1 if str(x).startswith(('10.', '172.16.', '192.168.'))␣
 ↪else 0
        )

    # Add more features from existing columns
    if 'dest_port' in features_df.columns:
        # Common ports
        features_df['is_web_port'] = features_df['dest_port'].apply(
            lambda x: 1 if x in [80, 443, 8080, 8443] else 0
        )
```

```python
        features_df['is_mail_port'] = features_df['dest_port'].apply(
            lambda x: 1 if x in [25, 465, 587, 110, 143, 993, 995] else 0
        )
        features_df['is_database_port'] = features_df['dest_port'].apply(
            lambda x: 1 if x in [1433, 3306, 5432, 27017, 6379, 9200] else 0
        )

    # Add protocol encoding
    if 'protocol' in features_df.columns:
        protocol_dummies = pd.get_dummies(features_df['protocol'],␣
↪prefix='protocol')
        features_df = pd.concat([features_df, protocol_dummies], axis=1)

    # Add country encoding
    if 'country' in features_df.columns:
        # Get top 10 countries and create dummies
        top_countries = features_df['country'].value_counts().nlargest(10).index
        features_df['country_encoded'] = features_df['country'].apply(
            lambda x: x if x in top_countries else 'Other'
        )
        country_dummies = pd.get_dummies(features_df['country_encoded'],␣
↪prefix='country')
        features_df = pd.concat([features_df, country_dummies], axis=1)

    # Add bytes features
    if 'bytes' in features_df.columns:
        features_df['bytes_log'] = features_df['bytes'].apply(
            lambda x: np.log1p(x) if x and not pd.isna(x) else 0
        )
        features_df['large_transfer'] = features_df['bytes'].apply(
            lambda x: 1 if x and not pd.isna(x) and x > 1000000 else 0
        )

    return features_df

# Apply feature engineering
features_df = extract_features(df)
features_df.head()
```

[15]:

```
   event_id               timestamp       source_ip  source_port  \
0         1 2025-03-28 00:59:59.649   35.203.210.32        50930
1         2 2025-03-28 00:59:59.649   35.203.210.32        50930
2         3 2025-03-28 00:59:58.402  38.132.109.168        57070
3         4 2025-03-28 00:59:58.402  38.132.109.168        57070
4         5 2025-03-28 00:59:56.032    213.32.32.91        33270

         dest_ip  dest_port             action protocol   bytes  \
```

```
0  195.201.244.27      47466  chs add blocker 3      tcp     44
1  195.201.244.27      47466  chs dropped input      tcp     44
2  188.40.207.210       1723  chs dropped input      tcp     40
3  188.40.207.210       1723  chs add blocker 3      tcp     40
4  188.40.207.214       3400  chs add blocker 3      tcp     60

          country  … country_Germany country_Hong Kong country_India  \
0  United Kingdom  …           False             False         False
1  United Kingdom  …           False             False         False
2   United States  …           False             False         False
3   United States  …           False             False         False
4          France  …           False             False         False

   country_Other  country_The Netherlands  country_United Kingdom  \
0          False                    False                    True
1          False                    False                    True
2          False                    False                   False
3          False                    False                   False
4          False                    False                   False

   country_United States  country_Vietnam  bytes_log  large_transfer
0                  False            False   3.806662               0
1                  False            False   3.806662               0
2                   True            False   3.713572               0
3                   True            False   3.713572               0
4                  False            False   4.110874               0

[5 rows x 41 columns]
```

[16]:
```python
# Select numerical features for clustering
numerical_features = features_df.select_dtypes(include=['int64', 'float64']).
 ↪columns.tolist()
print("Numerical features for clustering:")
print(numerical_features)

# Create feature matrix
X = features_df[numerical_features].copy()

# Handle missing values
X = X.fillna(0)

# Normalize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

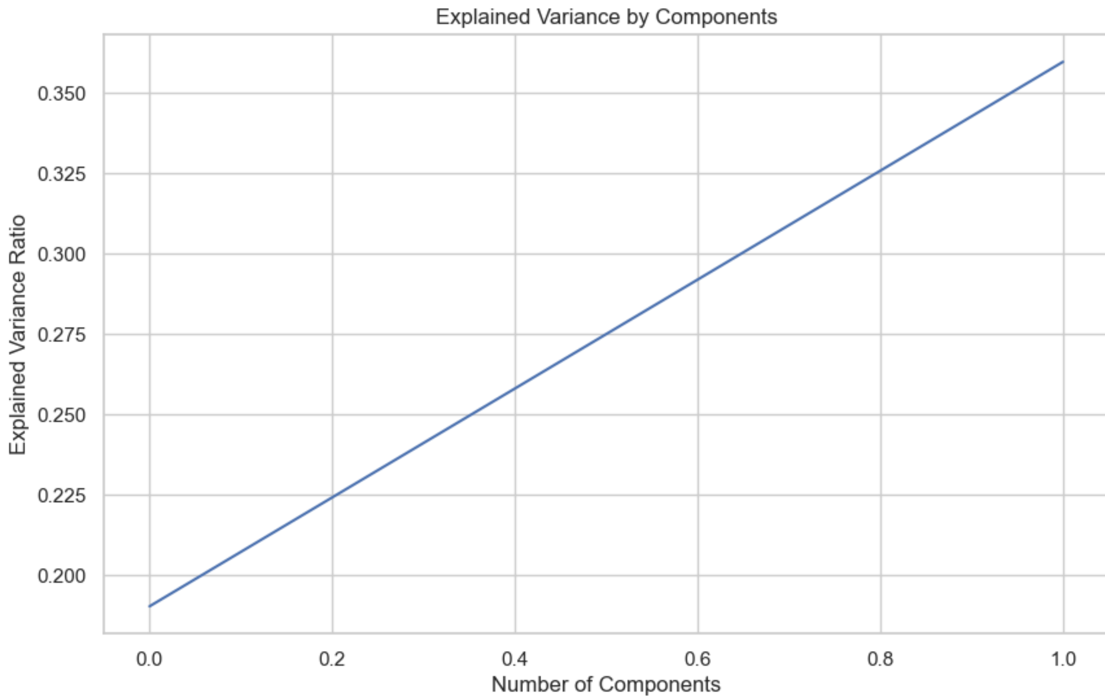
```
Numerical features for clustering:
['is_weekend', 'is_business_hours', 'source_ip_is_private', 'is_web_port',
'is_mail_port', 'is_database_port', 'bytes_log', 'large_transfer']
```

1.3 3. Dimensionality Reduction

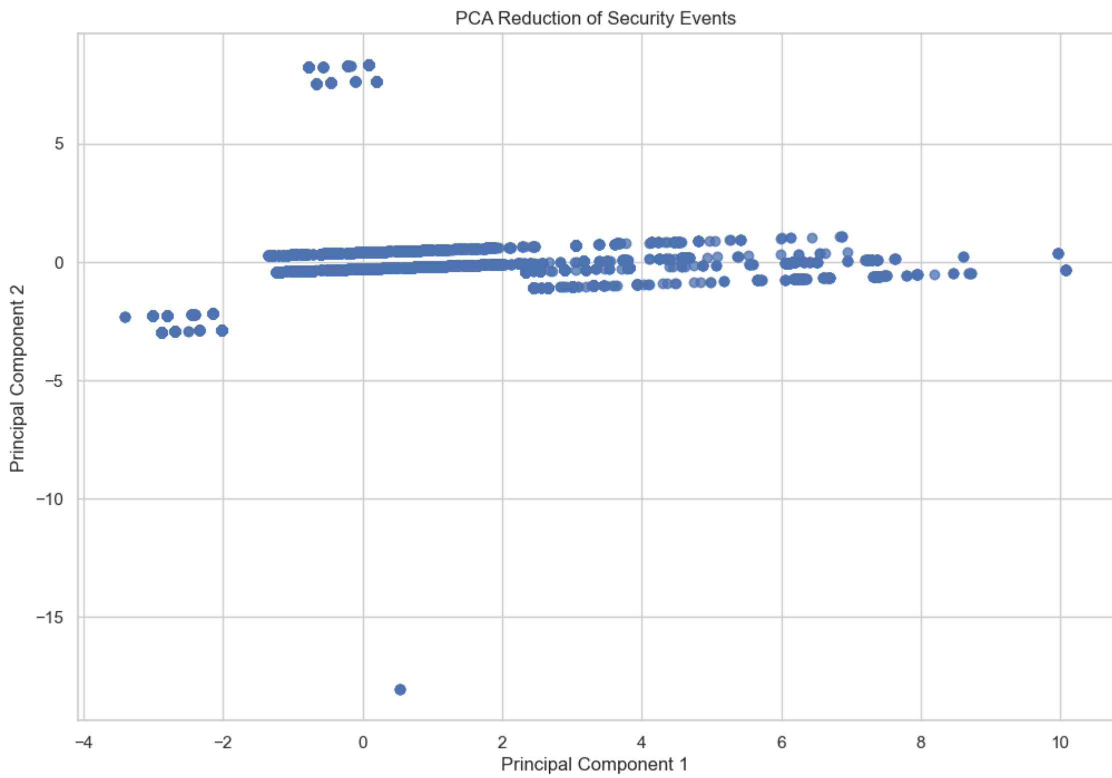
We'll use PCA to reduce dimensions before clustering.

[17]:
```
# Check if we have enough features for PCA
print(f"Dataset shape: {X_scaled.shape}")

# At least 3 features needed for meaningful dimensionality reduction
if X_scaled.shape[1] < 3:
    print("Not enough features for PCA, creating synthetic features...")

    # Create some additional synthetic features if needed
    if 'source_port' in features_df.columns and 'dest_port' in features_df.
 ↪columns:
        X = np.column_stack([
            X,
            features_df['source_port'].fillna(0).values % 1000,  # Port modulo
            features_df['dest_port'].fillna(0).values % 1000     # Port modulo
        ])
        X_scaled = StandardScaler().fit_transform(X)

    print(f"New shape after adding synthetic features: {X_scaled.shape}")

# Use PCA with at most n_components = min(n_samples, n_features) - 1
n_components = min(2, min(X_scaled.shape[0], X_scaled.shape[1]) - 1)
print(f"Using PCA with {n_components} components")

pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X_scaled)

# Plot the explained variance if we have enough components
if n_components > 1:
    plt.figure(figsize=(10, 6))
    plt.plot(np.cumsum(pca.explained_variance_ratio_))
    plt.xlabel('Number of Components')
    plt.ylabel('Explained Variance Ratio')
    plt.title('Explained Variance by Components')
    plt.grid(True)
    plt.show()
```

```
Dataset shape: (100000, 8)
Using PCA with 2 components
```



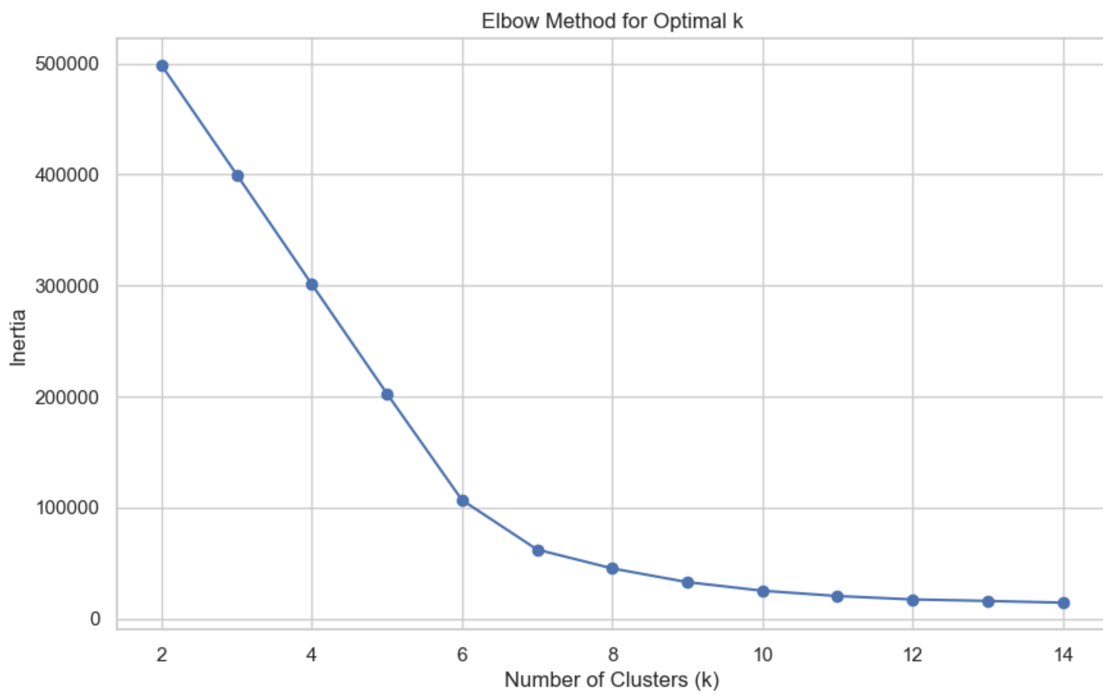
[18]:
```
# Handle the case where we only have 1D data
if X_pca.shape[1] == 1:
    # Create a simple 2D visualization with a random jitter on the y-axis
    plt.figure(figsize=(12, 8))
    plt.scatter(X_pca[:, 0], np.random.normal(0, 0.01, size=X_pca.shape[0]),␣
↪alpha=0.5)
    plt.title('PCA Reduction of Security Events (1D with jitter)')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Random Jitter')
    plt.show()
else:
    # Regular 2D PCA plot
    plt.figure(figsize=(12, 8))
    plt.scatter(X_pca[:, 0], X_pca[:, 1], alpha=0.5)
    plt.title('PCA Reduction of Security Events')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.show()
```



1.4 4. K-Means Clustering

Let's use K-Means to identify clusters in our security data.

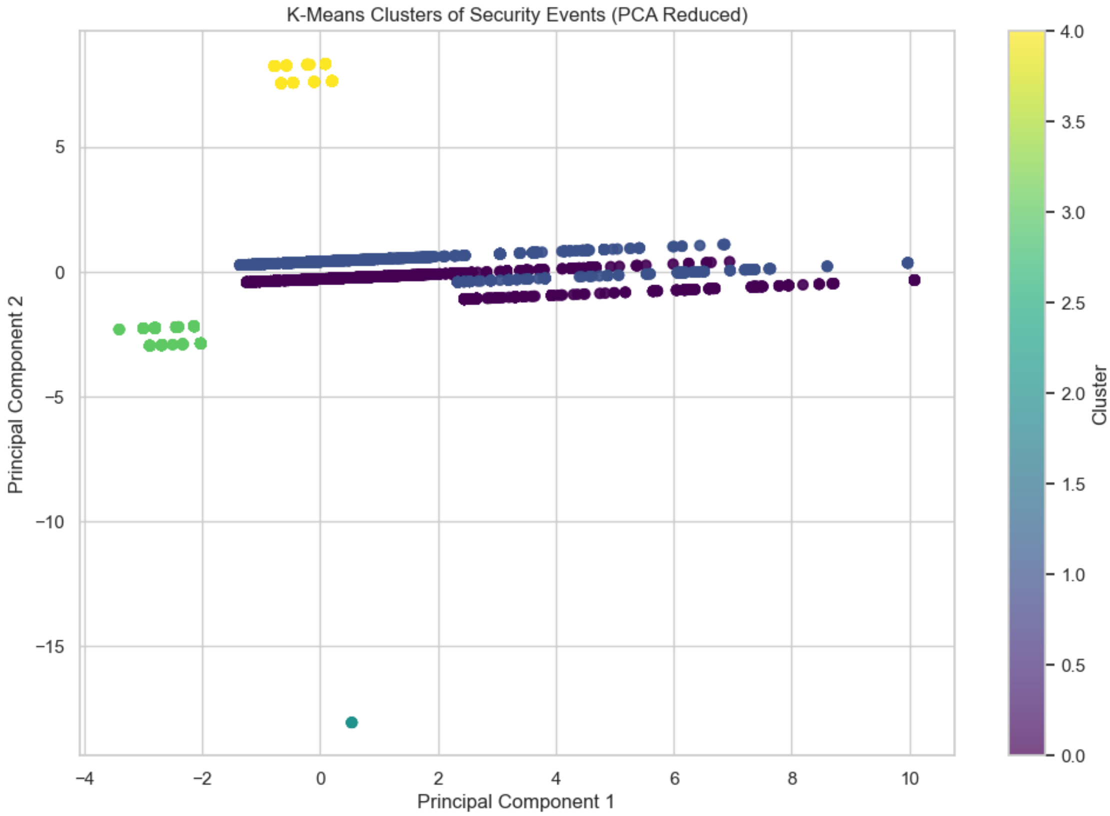
[19]:
```
# Determine optimal number of clusters using the Elbow method
inertia = []
k_range = range(2, 15)

for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

# Plot the elbow curve
plt.figure(figsize=(10, 6))
plt.plot(k_range, inertia, 'o-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal k')
plt.grid(True)
plt.show()
```



[20]:
```python
# Choose the number of clusters based on the elbow plot
optimal_k = 5  # This should be adjusted based on the elbow plot

# Apply K-Means clustering
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X_scaled)

# Add cluster labels to our dataframe
features_df['cluster'] = clusters
```

[21]:
```python
# Visualize the clusters in PCA space
plt.figure(figsize=(12, 8))

# Handle the case where we only have 1D data
if X_pca.shape[1] == 1:
    # Use the PCA component and add random jitter for visualization
    jitter = np.random.normal(0, 0.01, size=X_pca.shape[0])
    scatter = plt.scatter(X_pca[:, 0], jitter, c=clusters, cmap='viridis',␣
 ↪alpha=0.7)
    plt.title('K-Means Clusters of Security Events (PCA Reduced with jitter)')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Random Jitter')
else:
    # Regular 2D visualization
```

```
    scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='viridis',␣
↪alpha=0.7)
    plt.title('K-Means Clusters of Security Events (PCA Reduced)')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')

plt.colorbar(scatter, label='Cluster')
plt.show()
```



1.5 5. DBSCAN Clustering

Let's try DBSCAN as an alternative clustering method, which can find clusters of arbitrary shape and identify outliers.

[22]:
```
# Apply DBSCAN clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_clusters = dbscan.fit_predict(X_scaled)

# Add DBSCAN cluster labels to our dataframe
features_df['dbscan_cluster'] = dbscan_clusters
```

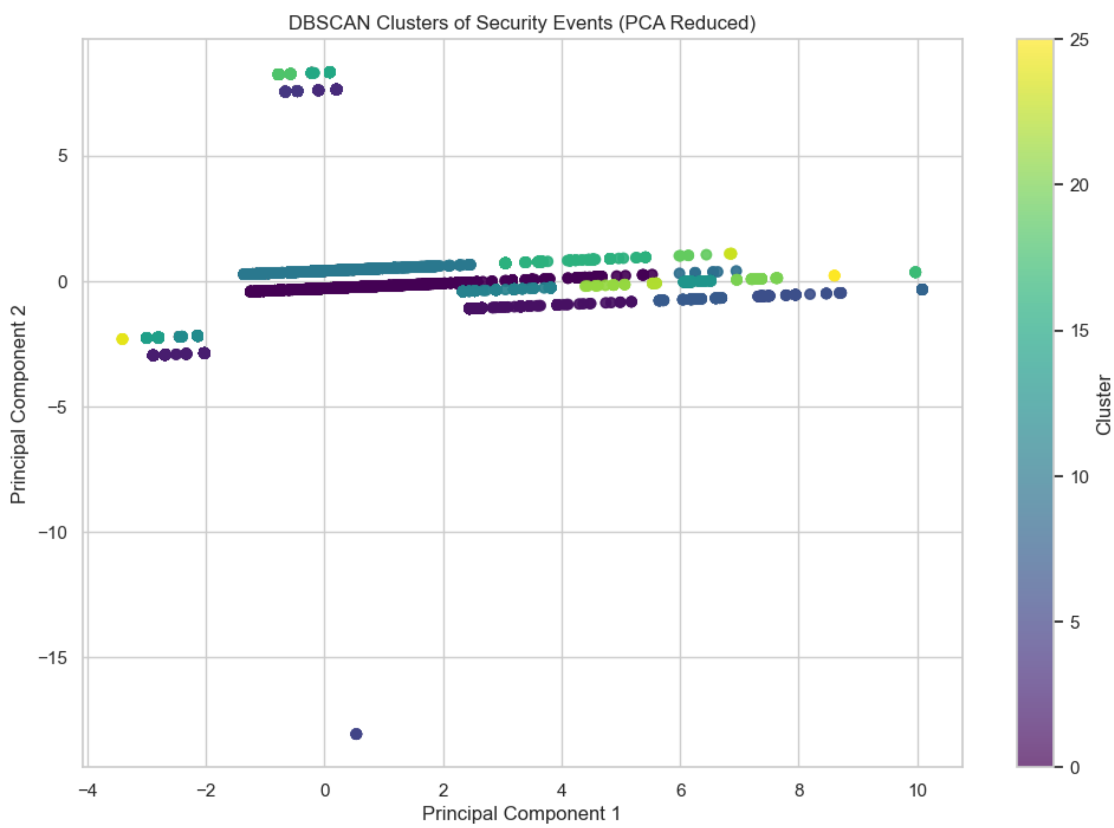
```
# Count occurrences of each cluster
print("DBSCAN Cluster Distribution:")
print(pd.Series(dbscan_clusters).value_counts().sort_index())
print(f"\nNumber of noise points (cluster -1): {sum(dbscan_clusters == -1)}")
```

```
DBSCAN Cluster Distribution:
0     52367
1      2737
2       920
3       376
4       234
5         9
6       132
7       265
8        26
9        14
10    38883
11     2061
12      204
13      144
14      580
15      334
16      344
17       14
18      204
19       10
20       77
21       30
22       13
23        8
24        8
25        6
Name: count, dtype: int64

Number of noise points (cluster -1): 0
```

[23]:

```
# Visualize DBSCAN clusters
plt.figure(figsize=(12, 8))

# Handle the case where we only have 1D data
if X_pca.shape[1] == 1:
    # Use the PCA component and add random jitter for visualization
    jitter = np.random.normal(0, 0.01, size=X_pca.shape[0])
    scatter = plt.scatter(X_pca[:, 0], jitter, c=dbscan_clusters,␣
 ↪cmap='viridis', alpha=0.7)
    plt.title('DBSCAN Clusters of Security Events (PCA Reduced with jitter)')
    plt.xlabel('Principal Component 1')
```

```
    plt.ylabel('Random Jitter')
else:
    # Regular 2D visualization
    scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=dbscan_clusters,␣
↪cmap='viridis', alpha=0.7)
    plt.title('DBSCAN Clusters of Security Events (PCA Reduced)')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')

plt.colorbar(scatter, label='Cluster')
plt.show()
```



1.6 6. Cluster Analysis

Let's analyze the characteristics of each cluster to identify potential attack patterns.

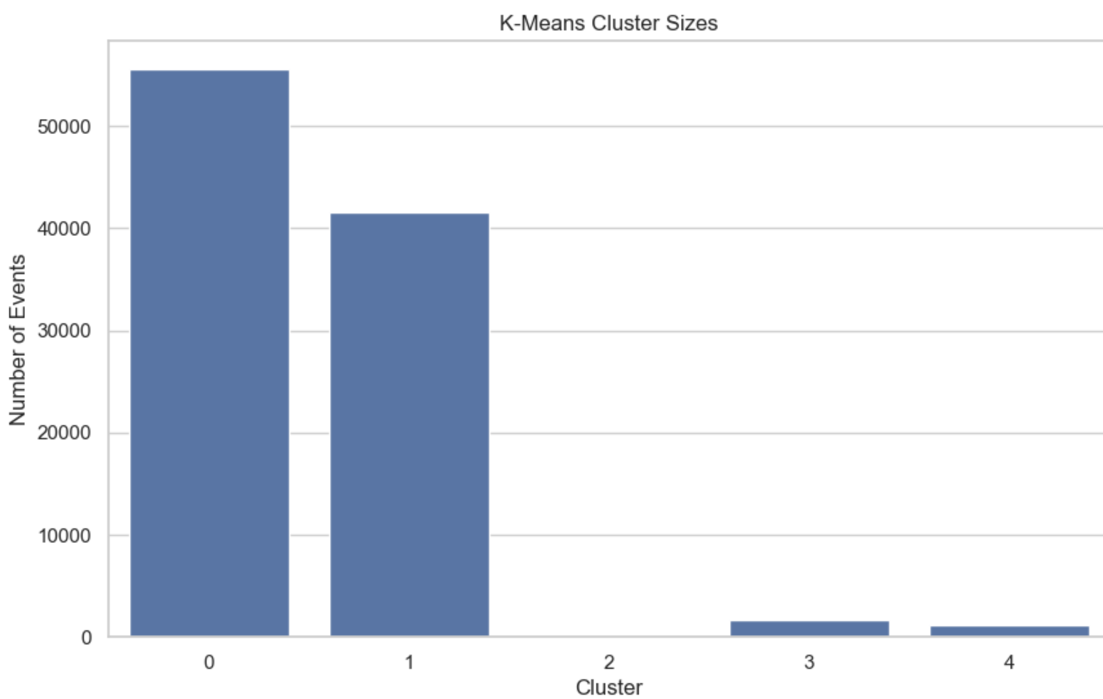
[24]:
```
# Analyze K-Means clusters
cluster_stats = features_df.groupby('cluster').agg({
    'action': lambda x: x.value_counts().index[0] if 'action' in features_df.
↪columns else None,  # Most common action
```

```
    'source_ip': 'nunique',                             # Number of unique␣
↪source IPs
    'hour_of_day': 'mean',                              # Average hour of day
    'source_ip_is_private': 'mean'                      # Proportion of private␣
↪IPs
}).reset_index()

print("K-Means Cluster Characteristics:")
display(cluster_stats)
```

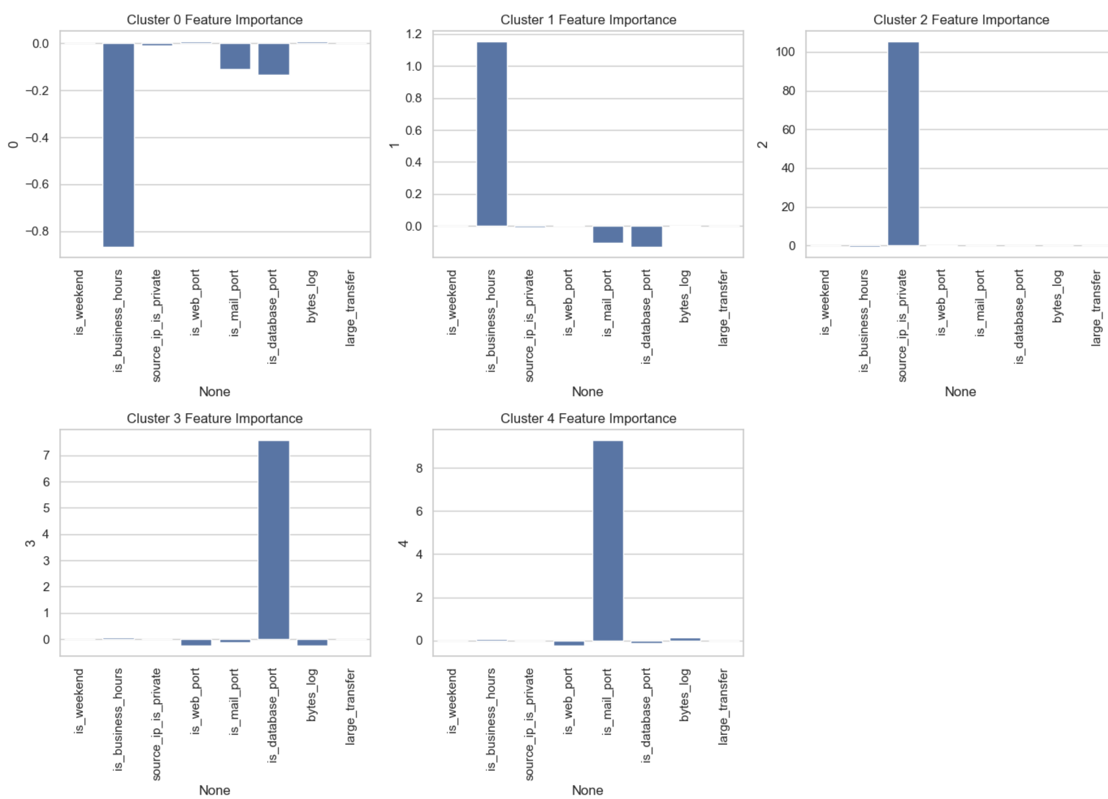
```
K-Means Cluster Characteristics:
```

	cluster	action	source_ip	hour_of_day	source_ip_is_private
0	0	chs add blocker 3	9141	4.056589	0.0
1	1	chs dropped input	6946	12.127314	0.0
2	2	chs dropped vlan28	1	0.000000	1.0
3	3	chs add blocker 3	484	7.747664	0.0
4	4	chs dropped input	312	7.945993	0.0

[25]:
```
# Visualize cluster size distribution
plt.figure(figsize=(10, 6))
cluster_counts = features_df['cluster'].value_counts().sort_index()
sns.barplot(x=cluster_counts.index, y=cluster_counts.values)
plt.title('K-Means Cluster Sizes')
plt.xlabel('Cluster')
plt.ylabel('Number of Events')
plt.show()
```



[26]:
```python
# Feature importance for each cluster
# Calculate feature means for each cluster
cluster_centers = pd.DataFrame(kmeans.cluster_centers_, columns=X.columns)

# Visualize feature importance for each cluster
plt.figure(figsize=(14, 10))
for i in range(optimal_k):
    plt.subplot(2, 3, i+1)
    sns.barplot(x=cluster_centers.columns, y=cluster_centers.iloc[i])
    plt.title(f'Cluster {i} Feature Importance')
    plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



1.7 7. Identifying Potential Attack Patterns

Now let's analyze each cluster to identify potential attack patterns.

[27]:
```python
# Analyze action types within each cluster
for cluster_id in sorted(features_df['cluster'].unique()):
    cluster_data = features_df[features_df['cluster'] == cluster_id]
    print(f"\nCluster {cluster_id} - Total Events: {len(cluster_data)}")

    # Action type distribution
    if 'action' in cluster_data.columns:
        print("Action type distribution:")
        print(cluster_data['action'].value_counts())

    # Protocol distribution
    if 'protocol' in cluster_data.columns:
        print("\nProtocol distribution:")
        print(cluster_data['protocol'].value_counts())

    # Source IP analysis
    if 'source_ip' in cluster_data.columns:
        print("\nTop 5 source IPs:")
        print(cluster_data['source_ip'].value_counts().head(5))

    # Time-based patterns
    if 'hour_of_day' in cluster_data.columns:
        print(f"\nAverage hour of day: {cluster_data['hour_of_day'].mean():.
↪2f}")

    print("-" * 50)
```

```
Cluster 0 - Total Events: 55541
Action type distribution:
action
chs add blocker 3      27472
chs dropped input      27472
chs add blocker 180      522
chs add blocker 1         75
Name: count, dtype: int64

Protocol distribution:
protocol
tcp     49988
udp      4940
icmp      557
41         28
47         18
4          10
Name: count, dtype: int64

Top 5 source IPs:
```

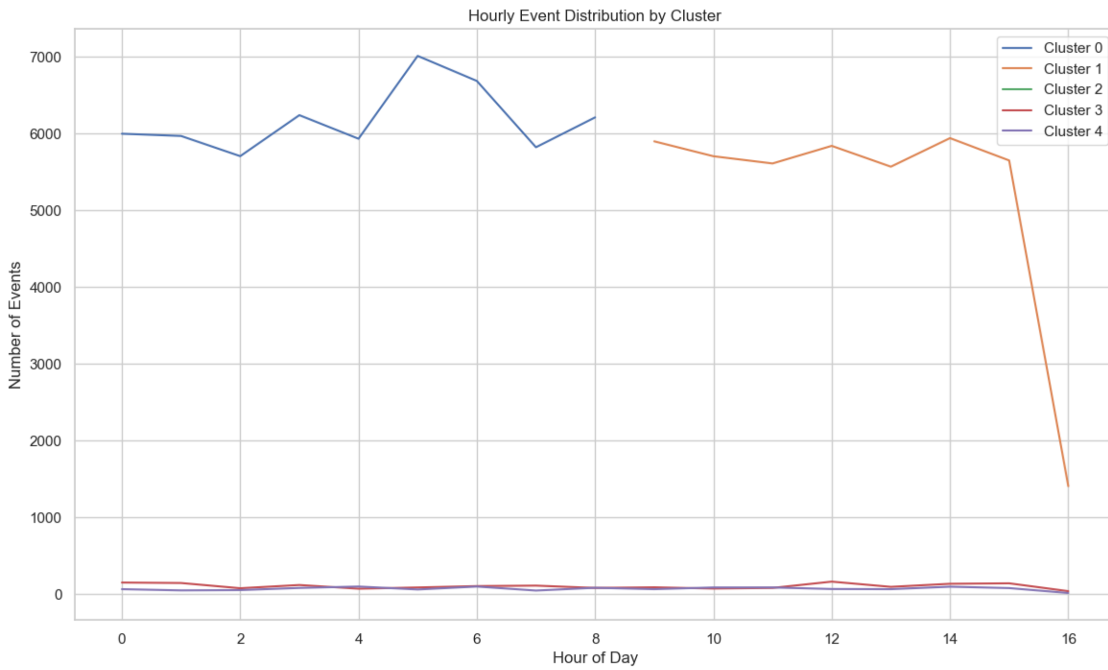
```
source_ip
104.156.155.10    348
79.124.62.122     322
79.124.62.134     314
134.209.173.54    308
79.124.62.126     300
Name: count, dtype: int64

Average hour of day: 4.06
--------------------------------------------------

Cluster 1 - Total Events: 41590
Action type distribution:
action
chs dropped input      20616
chs add blocker 3      20615
chs add blocker 180      317
chs add blocker 1         42
Name: count, dtype: int64

Protocol distribution:
protocol
tcp     37744
udp      3388
icmp      424
47         18
4          16
Name: count, dtype: int64

Top 5 source IPs:
source_ip
134.209.173.54    268
79.124.62.126     262
185.91.127.81     214
83.222.190.254    204
204.76.203.80     202
Name: count, dtype: int64

Average hour of day: 12.13
--------------------------------------------------

Cluster 2 - Total Events: 9
Action type distribution:
action
chs dropped vlan28    9
Name: count, dtype: int64

Protocol distribution:
```

```
protocol
tcp    9
Name: count, dtype: int64

Top 5 source IPs:
source_ip
10.5.28.140    9
Name: count, dtype: int64

Average hour of day: 0.00
--------------------------------------------------

Cluster 3 - Total Events: 1712
Action type distribution:
action
chs add blocker 3    856
chs dropped input    856
Name: count, dtype: int64

Protocol distribution:
protocol
tcp    1690
udp      22
Name: count, dtype: int64

Top 5 source IPs:
source_ip
80.82.70.133       28
14.241.229.29      26
119.201.111.206    22
142.93.58.55       22
190.129.65.235     20
Name: count, dtype: int64

Average hour of day: 7.75
--------------------------------------------------

Cluster 4 - Total Events: 1148
Action type distribution:
action
chs dropped input    574
chs add blocker 3    574
Name: count, dtype: int64

Protocol distribution:
protocol
tcp    1138
udp      10
```

```
Name: count, dtype: int64

Top 5 source IPs:
source_ip
185.93.89.48       200
194.0.234.11        86
103.219.169.232     26
45.82.78.100        18
195.211.191.226     18
Name: count, dtype: int64

Average hour of day: 7.95
--------------------------------------------------
```

[28]:

```
# Visualize hourly patterns by cluster
plt.figure(figsize=(14, 8))
for cluster_id in sorted(features_df['cluster'].unique()):
    cluster_data = features_df[features_df['cluster'] == cluster_id]
    if 'hour_of_day' in cluster_data.columns:
        hour_counts = cluster_data['hour_of_day'].value_counts().sort_index()
        plt.plot(hour_counts.index, hour_counts.values, label=f'Cluster␣
↪{cluster_id}')

plt.title('Hourly Event Distribution by Cluster')
plt.xlabel('Hour of Day')
plt.ylabel('Number of Events')
plt.legend()
plt.grid(True)
plt.show()
```



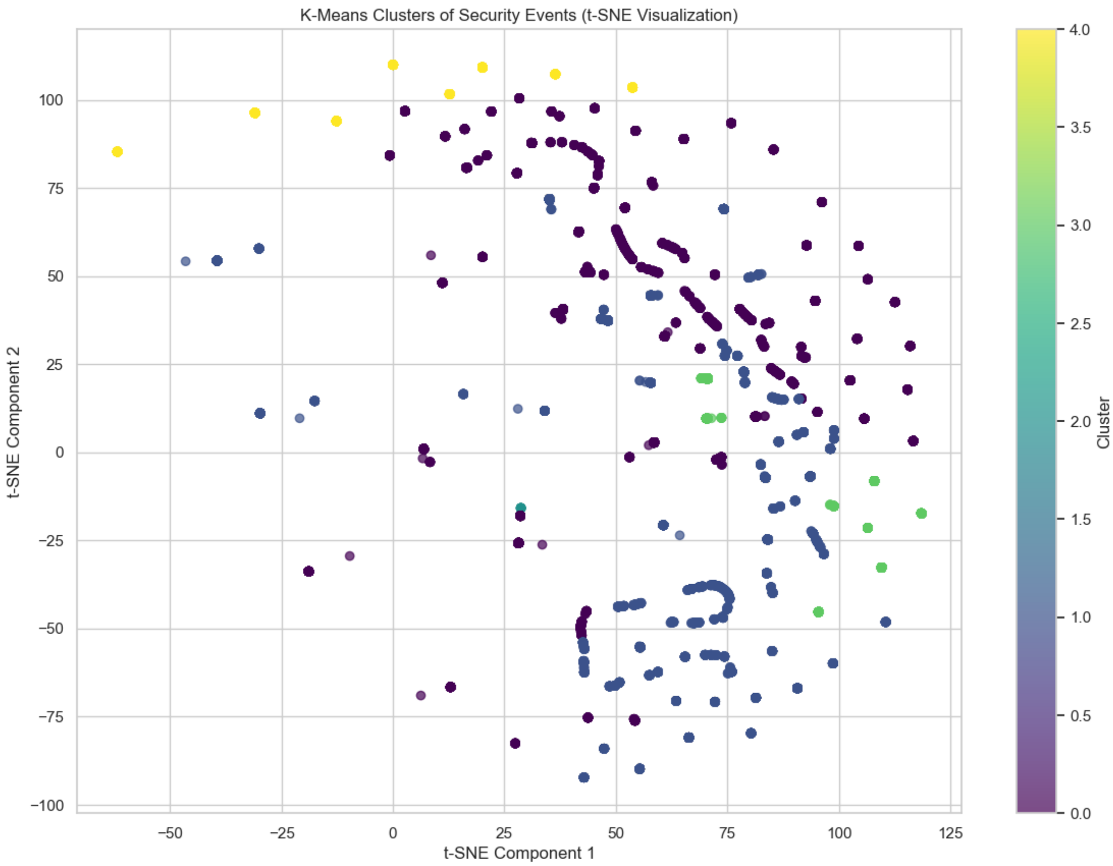
1.8 8. Advanced Visualization with t-SNE

Let's use t-SNE to create a more detailed visualization of our security event clusters.

[33]:
```
# Skip t-SNE if we don't have enough features
if X_scaled.shape[1] < 3:
    print("Skipping t-SNE visualization as we don't have enough features")
    # Create synthetic data for visualization
    X_tsne = np.column_stack([
        X_pca.flatten(),
        np.random.normal(0, 0.1, size=X_pca.shape[0])
    ])
else:
    # Apply t-SNE for better visualization
    tsne = TSNE(n_components=2, random_state=42, perplexity=min(30, X_scaled.
  ↪shape[0]//5), max_iter=1000)
    X_tsne = tsne.fit_transform(X_scaled)
```

[30]:
```
# Visualize K-Means clusters with t-SNE
plt.figure(figsize=(14, 10))
scatter = plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=clusters, cmap='viridis',␣
  ↪alpha=0.7)
plt.colorbar(scatter, label='Cluster')
plt.title('K-Means Clusters of Security Events (t-SNE Visualization)')
plt.xlabel('t-SNE Component 1')
```

```
plt.ylabel('t-SNE Component 2')
plt.show()
```



1.9 9. Identifying Attack Patterns

Based on our cluster analysis, we can identify potential attack patterns or anomalous behaviors.

[31]:
```
# Identify potential attack clusters based on characteristics
def identify_suspicious_clusters(features_df, cluster_col='cluster'):
    suspicious_clusters = []

    for cluster_id in sorted(features_df[cluster_col].unique()):
        cluster_data = features_df[features_df[cluster_col] == cluster_id]

        # Criteria for suspicious clusters (these are examples - adjust based
 ↪on your data)
        criteria = []

        # 1. High proportion of specific actions (if available)
        if 'action' in cluster_data.columns:
```

```
            # Look for specific actions that might indicate malicious activity
            # For example, if 'block' is an action
            if 'block' in cluster_data['action'].values:
                block_rate = (cluster_data['action'] == 'block').mean()
                if block_rate > 0.7:  # Over 70% blocked
                    criteria.append(f"High block rate: {block_rate:.2f}")

        # 2. Events concentrated in unusual hours (e.g., midnight to 5 AM)
        if 'hour_of_day' in cluster_data.columns:
            night_hours = (cluster_data['hour_of_day'] >= 0) &
↪(cluster_data['hour_of_day'] < 5)
            night_rate = night_hours.mean()
            if night_rate > 0.5:  # Over 50% at night
                criteria.append(f"Unusual timing: {night_rate:.2f} activity
↪during midnight-5AM")

        # 3. High number of unique source IPs accessing same target
        if 'source_ip' in cluster_data.columns and 'dest_ip' in cluster_data.
↪columns:
            # Group by destination IP and count unique source IPs
            ip_counts = cluster_data.groupby('dest_ip')['source_ip'].nunique()
            if any(ip_counts > 10):  # More than 10 source IPs targeting same
↪destination
                criteria.append(f"Multiple sources targeting same destination:
↪max {ip_counts.max()} sources")

        # 4. Suspicious countries (if available)
        if 'country' in cluster_data.columns:
            country_counts = cluster_data['country'].value_counts()
            suspicious_countries = ['Unknown', 'Russia', 'North Korea', 'Iran',
↪'China']
            for country in suspicious_countries:
                if country in country_counts.index:
                    country_rate = country_counts[country] / len(cluster_data)
                    if country_rate > 0.3:  # Over 30% from suspicious country
                        criteria.append(f"High traffic from {country}:
↪{country_rate:.2f}")

        # If any criteria are met, consider this cluster suspicious
        if criteria:
            suspicious_clusters.append({
                'cluster_id': cluster_id,
                'size': len(cluster_data),
                'reasons': criteria
            })
```

```
    return suspicious_clusters

# Apply the function to identify suspicious clusters
suspicious_clusters = identify_suspicious_clusters(features_df)

# Display the results
print("Potentially Suspicious Clusters:")
for cluster in suspicious_clusters:
    print(f"\nCluster {cluster['cluster_id']} (Size: {cluster['size']})")
    print("Suspicious characteristics:")
    for reason in cluster['reasons']:
        print(f"- {reason}")
```

```
Potentially Suspicious Clusters:

Cluster 0 (Size: 55541)
Suspicious characteristics:
- Unusual timing: 0.54 activity during midnight-5AM
- Multiple sources targeting same destination: max 2120 sources

Cluster 1 (Size: 41590)
Suspicious characteristics:
- Multiple sources targeting same destination: max 1572 sources

Cluster 2 (Size: 9)
Suspicious characteristics:
- Unusual timing: 1.00 activity during midnight-5AM

Cluster 3 (Size: 1712)
Suspicious characteristics:
- Multiple sources targeting same destination: max 83 sources

Cluster 4 (Size: 1148)
Suspicious characteristics:
- Multiple sources targeting same destination: max 39 sources
```

1.10 10. Recommendations for Security Monitoring

Based on our clustering analysis, here are some recommendations for security monitoring and incident response.

[32]:
```
# Extract events from suspicious clusters for further investigation
suspicious_events = pd.DataFrame()

for cluster in suspicious_clusters:
    cluster_id = cluster['cluster_id']
    cluster_events = features_df[features_df['cluster'] == cluster_id].copy()
    cluster_events['suspicious_reason'] = ', '.join(cluster['reasons'])
```

```python
    suspicious_events = pd.concat([suspicious_events, cluster_events])

# Save suspicious events for investigation
if not suspicious_events.empty:
    # Display sample of suspicious events
    print(f"Total suspicious events identified: {len(suspicious_events)}")
    print("\nSample of suspicious events:")
    display(suspicious_events.head())

    # Consider saving to a file for further investigation
    # suspicious_events.to_csv('../results/suspicious_events.csv', index=False)
```

```
Total suspicious events identified: 100000

Sample of suspicious events:

   event_id               timestamp       source_ip  source_port  \
0         1 2025-03-28 00:59:59.649   35.203.210.32        50930
1         2 2025-03-28 00:59:59.649   35.203.210.32        50930
2         3 2025-03-28 00:59:58.402  38.132.109.168        57070
3         4 2025-03-28 00:59:58.402  38.132.109.168        57070
4         5 2025-03-28 00:59:56.032    213.32.32.91        33270

          dest_ip  dest_port             action protocol  bytes  \
0  195.201.244.27      47466  chs add blocker 3      tcp     44
1  195.201.244.27      47466  chs dropped input      tcp     44
2  188.40.207.210       1723  chs dropped input      tcp     40
3  188.40.207.210       1723  chs add blocker 3      tcp     40
4  188.40.207.214       3400  chs add blocker 3      tcp     60

          country  … country_Other country_The Netherlands  \
0  United Kingdom  …         False                   False
1  United Kingdom  …         False                   False
2   United States  …         False                   False
3   United States  …         False                   False
4          France  …         False                   False

  country_United Kingdom  country_United States  country_Vietnam  bytes_log  \
0                   True                  False            False   3.806662
1                   True                  False            False   3.806662
2                  False                   True            False   3.713572
3                  False                   True            False   3.713572
4                  False                  False            False   4.110874

   large_transfer  cluster  dbscan_cluster  \
0               0        0               0
1               0        0               0
2               0        0               0
```

```
3                0        0                0
4                0        0                0

                                   suspicious_reason
0  Unusual timing: 0.54 activity during midnight-…
1  Unusual timing: 0.54 activity during midnight-…
2  Unusual timing: 0.54 activity during midnight-…
3  Unusual timing: 0.54 activity during midnight-…
4  Unusual timing: 0.54 activity during midnight-…

[5 rows x 44 columns]
```

1.11 11. Conclusion

In this notebook, we demonstrated how clustering techniques can be applied to security logs to identify potential attack patterns. We used:

1. K-Means clustering to identify distinct groups of security events
2. DBSCAN to find outliers and clusters of arbitrary shapes
3. Dimensionality reduction techniques (PCA and t-SNE) for visualization
4. Feature engineering to extract meaningful attributes from security logs

The identified clusters can help security analysts focus their investigation on potentially suspicious activities, improving the efficiency of security monitoring and incident response processes.

Next steps could include: - Creating automated alerts based on the identified patterns - Developing a classification model trained on these clusters - Integrating this analysis into a real-time monitoring system