Project Title: Suspicious Web Threat Interaction Analysis

1. Introduction

This report presents a cybersecurity analysis focused on identifying and understanding suspicious web threat interactions through network logs. The objective is to analyze key parameters such as IP addresses, countries, threat types, and session behaviors to uncover potential vulnerabilities and patterns in web traffic.

2. Tools Used

- Python (Pandas, Matplotlib, NetworkX)
- Google Colab
- Word (for reporting)

3. Dataset Overview

The dataset contains logs of suspicious web sessions with the following columns:

- `src_ip`: Source IP Address
- `dst_port`: Destination Port
- `src_ip_country_code`: Source Country
- `detection_types`: Type of Detected Threat
- `time`: Timestamp of the event
- `observation_name`, `source.name`, `session_duration`, etc.

4. Data Preprocessing

- Removed null values from `src_ip` and `dst_port`
- Converted `time` column to datetime format
- Extracted `only_date` column from timestamp
- Cleaned and stripped column names

-

5. Analysis & Visualizations

5.1 Suspicious Sessions by Country

Bar chart created to visualize the number of sessions per country based on `src_ip_country_code`.

5.2 Top 10 Source IPs

Bar chart of the most active IPs sending suspicious traffic.

5.3 Activity Over Time

Line chart showing how suspicious activity changed over different dates. Major spike observed on April 26, 2024.

5.4 Threat Type Distribution

Pie chart showing threat types. All entries were of one type: `waf_rule`.

5.5 Country-Wise Summary Table

Table showing:

- Total Suspicious Sessions
- Unique IPs per country
- % Share of each country

Top Country: United States (113 sessions, 40%)

5.6 Network Graph: Source IP to Port Mapping

Visual graph using NetworkX showing interactions between source IPs and destination ports.

6. Future Analysis Possibilities

- Integrate machine learning for anomaly detection
- Expand threat type tracking beyond `waf_rule`
- Real-time alert system with stream processing (Kafka, ELK)
- IP risk scoring using third-party intel

7. Conclusion & Recommendations

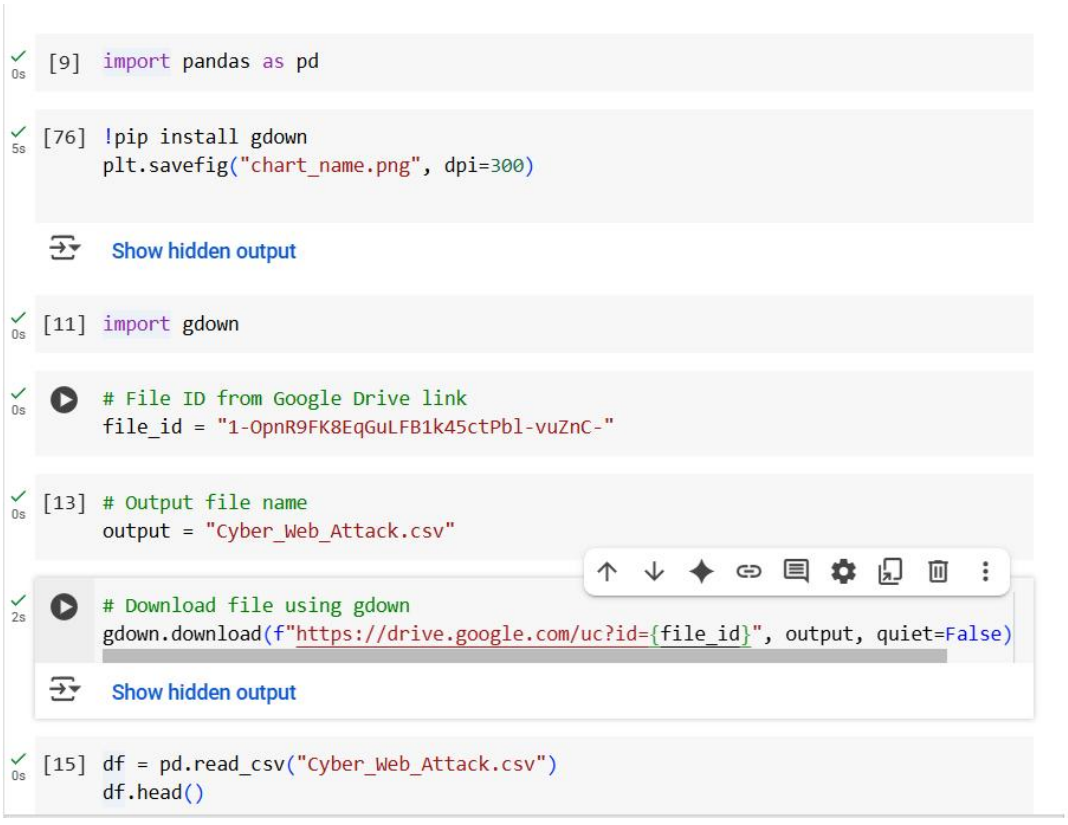
Summary:

- Highest threat source: US IPs
- Major spike in activity on 2024-04-26
- Single threat type detected (WAF Rule)

Recommendations:

- Improve logging coverage to detect more threat types
- Block or monitor high-risk IPs
- Setup alerts for time-based spikes

8. Screenshots Section



```
df = pd.read_csv("Cyber_Web_Attack.csv")
df.head()
```

	bytes_in	bytes_out	creation_time	end_time	src_ip	src_ip_country
0	5602	12990	2024-04-25T23:00:00Z	2024-04-25T23:10:00Z	147.161.161.82	
1	30912	18186	2024-04-25T23:00:00Z	2024-04-25T23:10:00Z	165.225.33.6	
2	28506	13468	2024-04-25T23:00:00Z	2024-04-25T23:10:00Z	165.225.212.255	
3	30546	14278	2024-04-25T23:00:00Z	2024-04-25T23:10:00Z	136.226.64.114	
4	6526	13892	2024-04-25T23:00:00Z	2024-04-25T23:10:00Z	165.225.240.79	

```
[16] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 282 entries, 0 to 281
Data columns (total 16 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   bytes_in             282 non-null    int64
 1   bytes_out            282 non-null    int64
 2   creation_time        282 non-null    object
 3   end_time             282 non-null    object
 4   src_ip               282 non-null    object
 5   src_ip_country_code  282 non-null    object
 6   protocol             282 non-null    object
 7   response.code        282 non-null    int64
 8   dst_port             282 non-null    int64
 9   dst_ip               282 non-null    object
 10  rule_names           282 non-null    object
 11  observation_name     282 non-null    object
 12  source.meta          282 non-null    object
 13  source.name          282 non-null    object
 14  time                 282 non-null    object
 15  detection_types      282 non-null    object
dtypes: int64(4), object(12)
memory usage: 35.4+ KB
```

```
df.isnull().sum()
```

	0
bytes_in	0
bytes_out	0
creation_time	0
end_time	0
src_ip	0
src_ip_country_code	0
protocol	0
response.code	0
dst_port	0
dst_ip	0
rule_names	0
observation_name	0
source.meta	0
source.name	0

```
[18] df.duplicated().sum()
```

```
np.int64(0)
```

```
[19] df.drop_duplicates(inplace=True)
```

```
[20] df.dtypes
```

	0
bytes_in	int64
bytes_out	int64
creation_time	object
end_time	object
src_ip	object
src_ip_country_code	object
protocol	object
response.code	int64

```
[21] df.columns
```

```
Index(['bytes_in', 'bytes_out', 'creation_time', 'end_time', 'src_ip',
       'src_ip_country_code', 'protocol', 'response.code', 'dst_port',
       'dst_ip', 'rule_names', 'observation_name', 'source.meta',
       'source.name', 'time', 'detection_types'],
      dtype='object')
```

```
[22] df['creation_time'] = pd.to_datetime(df['creation_time'])
```

```
[23] df['end_time'] = pd.to_datetime(df['end_time'])
```

```
[24] df['Year'] = df['creation_time'].dt.year
     df['Month'] = df['creation_time'].dt.month
     df['Day'] = df['creation_time'].dt.day
     df['Hour'] = df['creation_time'].dt.hour
     df['Weekday'] = df['creation_time'].dt.day_name()
```

```
[25] df.isnull().sum()
```

	0
bytes_in	0
bytes_out	0
creation_time	0
end_time	0
src_ip	0
src_ip_country_code	0
protocol	0
response.code	0
dst_port	0
dst_ip	0
rule_names	0
observation_name	0
source.meta	0

```
[26] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 282 entries, 0 to 281
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   bytes_in             282 non-null    int64
 1   bytes_out            282 non-null    int64
 2   creation_time        282 non-null    datetime64[ns, UTC]
 3   end_time             282 non-null    datetime64[ns, UTC]
 4   src_ip               282 non-null    object
 5   src_ip_country_code  282 non-null    object
 6   protocol             282 non-null    object
 7   response.code        282 non-null    int64
 8   dst_port             282 non-null    int64
 9   dst_ip               282 non-null    object
 10  rule_names           282 non-null    object
 11  observation_name     282 non-null    object
 12  source.meta          282 non-null    object
 13  source.name          282 non-null    object
 14  time                 282 non-null    object
 15  detection_types      282 non-null    object
 16  Year                 282 non-null    int32
 17  Month                282 non-null    int32
 18  Day                  282 non-null    int32
 19  Hour                 282 non-null    int32
 20  Weekday              282 non-null    object
dtypes: datetime64[ns, UTC](2), int32(4), int64(4), object(11)
```

```
[27] df.describe(include='all').T
```

	count	unique	top	freq	mean
bytes_in	282.0	NaN	NaN	NaN	1199390.191489
bytes_out	282.0	NaN	NaN	NaN	84554.29078
creation_time	282	NaN	NaN	NaN	2024-04-26 03:57:03.404255488+00:00
end_time	282	NaN	NaN	NaN	2024-04-26 04:07:03.404255232+00:00
src_ip	282	28	165.225.209.4	29	NaN
src_ip_country_code	282	7	US	113	NaN
protocol	282	1	HTTPS	282	NaN
response.code	282.0	NaN	NaN	NaN	200.0
dst_port	282.0	NaN	NaN	NaN	443.0
dst_ip	282	1	10.138.69.97	282	NaN
rule_names	282	1	Suspicious Web Traffic	282	NaN
observation_name	282	1	Adversary Infrastructure	282	NaN

```
[28] # 1. Calculate Session Duration in Seconds
     df['session_duration'] = (df['end_time'] - df['creation_time']).dt.total_seconds(
```

```
[29] # 2. Drop columns with only 1 unique value
     df.drop(columns=['dst_ip', 'rule_names', 'protocol'], inplace=True)
```

```
[30] # 3. Optional: Convert categorical columns (for visualizations)
     df['src_ip_country_code'] = df['src_ip_country_code'].astype('category')
     df['src_ip'] = df['src_ip'].astype('category')
```

```
[31] print(df.columns)
     print(df.dtypes)
     df[['session_duration']].head()
```

```
Index(['bytes_in', 'bytes_out', 'creation_time', 'end_time', 'src_ip',
       'src_ip_country_code', 'response.code', 'dst_port', 'observation_name',
       'source.meta', 'source.name', 'time', 'detection_types', 'Year',
       'Month', 'Day', 'Hour', 'Weekday', 'session_duration'],
      dtype='object')
bytes_in                             int64
bytes_out                            int64
creation_time          datetime64[ns, UTC]
end_time               datetime64[ns, UTC]
src_ip                            category
src_ip_country_code               category
```

```
[32] import seaborn as sns
     import matplotlib.pyplot as plt
```

```
[33] # Set figure size
     plt.figure(figsize=(10,6))
```

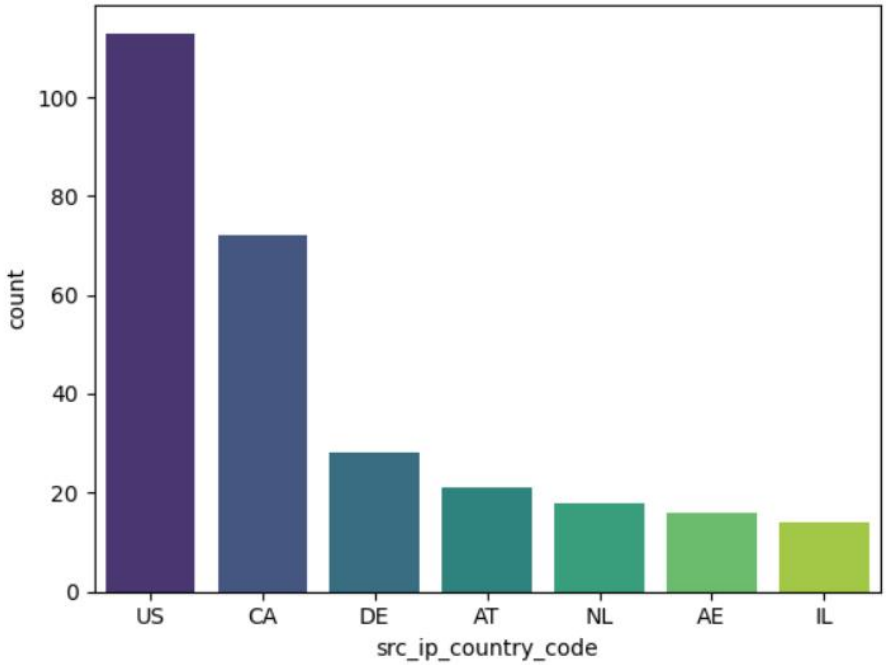
```
<Figure size 1000x600 with 0 Axes>
<Figure size 1000x600 with 0 Axes>
```

```
[34] # Plot
     sns.countplot(data=df, x='src_ip_country_code', order=df['src_ip_country_code'].v
```

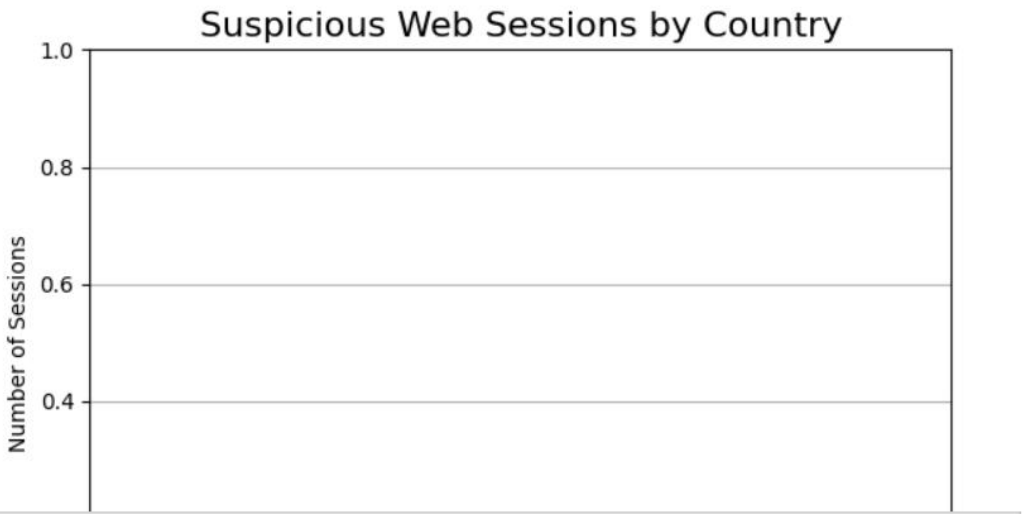
```
/tmp/ipython-input-34-4254722746.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.

  sns.countplot(data=df, x='src_ip_country_code', order=df['src_ip_country_code'].
<Axes: xlabel='src_ip_country_code', ylabel='count'>
```

```
  sns.countplot(data=df, x='src_ip_country_code', order=df['src_ip_country_code'].
```

```
<Axes: xlabel='src_ip_country_code', ylabel='count'>
```




```
[35] # Titles and labels
     plt.title("Suspicious Web Sessions by Country", fontsize=16)
     plt.xlabel("Country")
     plt.ylabel("Number of Sessions")
     plt.xticks(rotation=45)
     plt.grid(axis='y')
     plt.tight_layout()
     plt.show()
```

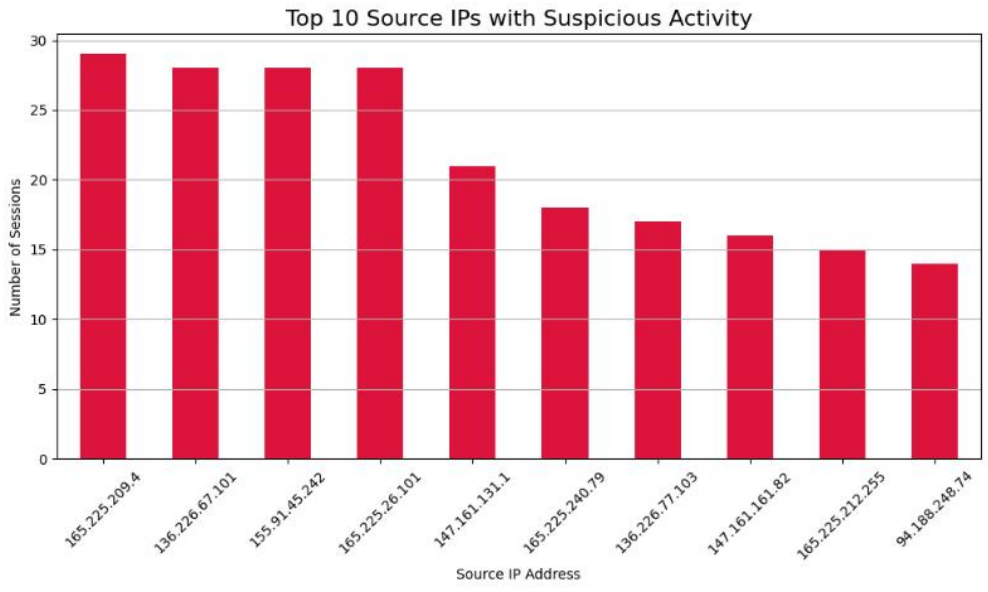


```
print(df.columns)
```

```
Index(['bytes_in', 'bytes_out', 'creation_time', 'end_time', 'src_ip',
       'src_ip_country_code', 'response.code', 'dst_port', 'observation_name',
       'source.meta', 'source.name', 'time', 'detection_types', 'Year',
       'Month', 'Day', 'Hour', 'Weekday', 'session_duration'],
      dtype='object')
```

```
[37] # Plot Top 10 source IPs by number of suspicious sessions
     top_src_ips = df['src_ip'].value_counts().head(10)
```

```
[38]
     plt.figure(figsize=(10, 6))
     top_src_ips.plot(kind='bar', color='crimson')
     plt.title("Top 10 Source IPs with Suspicious Activity", fontsize=16)
     plt.xlabel("Source IP Address")
     plt.ylabel("Number of Sessions")
     plt.xticks(rotation=45)
     plt.grid(axis='y')
     plt.tight_layout()
     plt.show()
```



```
[39] print(df['time'].head())
     print(df['time'].dtype)
```

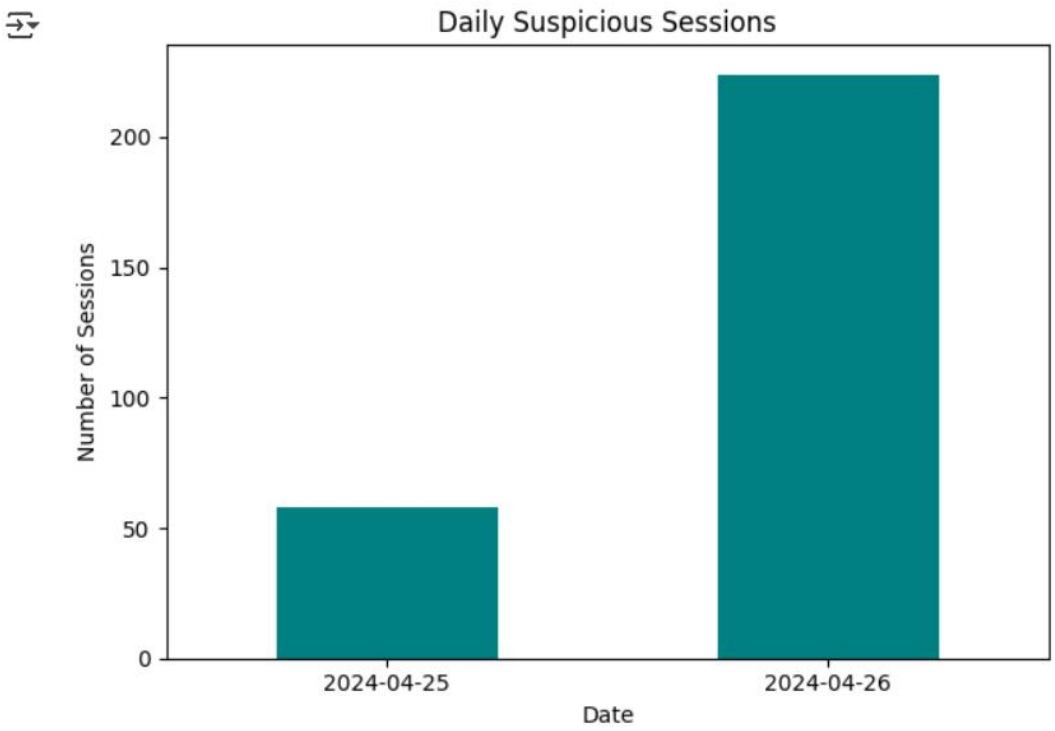
```
0    2024-04-25T23:00:00Z
1    2024-04-25T23:00:00Z
2    2024-04-25T23:00:00Z
3    2024-04-25T23:00:00Z
4    2024-04-25T23:00:00Z
Name: time, dtype: object
object
```

```
[40]
     # Convert to datetime (if not already)
     df['time'] = pd.to_datetime(df['time'])
```

```
[41] # Extract only date part for grouping
     df['only_date'] = df['time'].dt.date
```

```
[42] # Step 1: Count threat types
     threat_counts = df['detection_types'].value_counts()
```

```
[43] # Group by date and count sessions
     daily_sessions = df.groupby('only_date').size()
```




```
# Step 1: Group by Country
summary = df.groupby('src_ip_country_code').agg({
    'src_ip': 'nunique',        # Unique IPs
    'detection_types': 'count'  # Total Threats
}).rename(columns={
    'src_ip': 'Unique IPs',
    'detection_types': 'Suspicious Sessions'
})
```

```
/tmp/ipython-input-47-2399539546.py:2: FutureWarning: The default of observed=Fals
  summary = df.groupby('src_ip_country_code').agg({
```

```
[48] # Step 2: Add percentage column
summary['% Share'] = (summary['Suspicious Sessions'] / summary['Suspicious Sessic
```

```
[49] # Step 3: Sort descending
summary = summary.sort_values(by='Suspicious Sessions', ascending=False)
```

```
[50] # Show top 10
print(summary.head(10))
```

```
                     Unique IPs  Suspicious Sessions    % Share
src_ip_country_code
US                           19                  113  40.070922
CA                            4                   72  25.531915
DE                            1                   28   9.929078
```

```
[50] print(summary.head(10))
```

```
                        Unique IPs  Suspicious Sessions    % Share
src_ip_country_code
US                              19                  113  40.070922
CA                               4                   72  25.531915
DE                               1                   28   9.929078
AT                               1                   21   7.446809
NL                               1                   18   6.382979
AE                               1                   16   5.673759
IL                               1                   14   4.964539
```

```
[51] # Step 1: Check unique values first (optional)
     print(df['detection_types'].value_counts())
```

```
detection_types
waf_rule    282
Name: count, dtype: int64
```

```
[52] # Step 2: Clean null or empty values
     df_clean = df[df['detection_types'].notna() & (df['detection_types'] != '')]
```

```
[53] # Step 3: Count and plot
     threat_counts = df_clean['detection_types'].value_counts()
```


```
[51] # Step 1: Check unique values first (optional)
     print(df['detection_types'].value_counts())
```

```
detection_types
waf_rule    282
Name: count, dtype: int64
```

```
[52] # Step 2: Clean null or empty values
     df_clean = df[df['detection_types'].notna() & (df['detection_types'] != '')]
```

```
[53] # Step 3: Count and plot
     threat_counts = df_clean['detection_types'].value_counts()
```

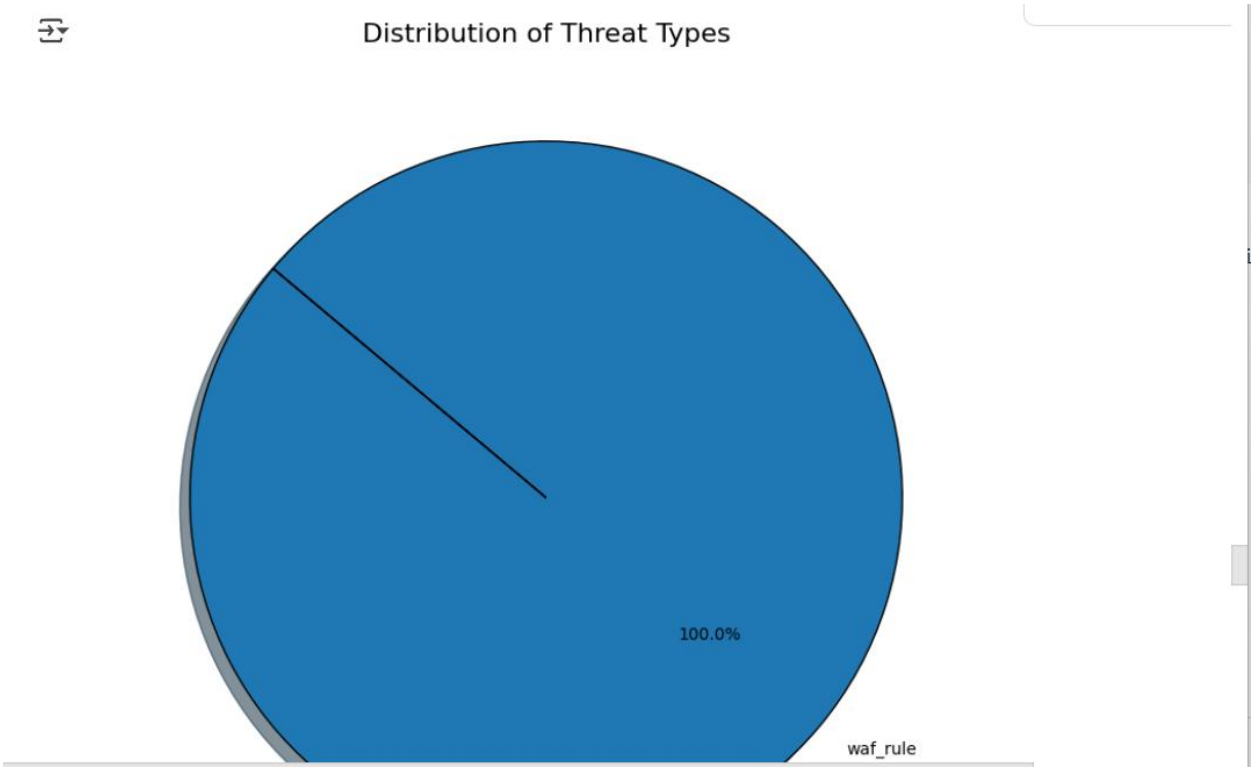

```
plt.figure(figsize=(8, 8))
PO*threat_counts.plot(
    kind='pie',
    autopct='%1.1f%%',
    startangle=140,
    shadow=True,
    wedgeprops={'linewidth': 1, 'edgecolor': 'black'}
)
plt.title("Distribution of Threat Types", fontsize=16)
plt.ylabel("")  # Hide y-label for cleaner pie
plt.tight_layout()
plt.show()
```

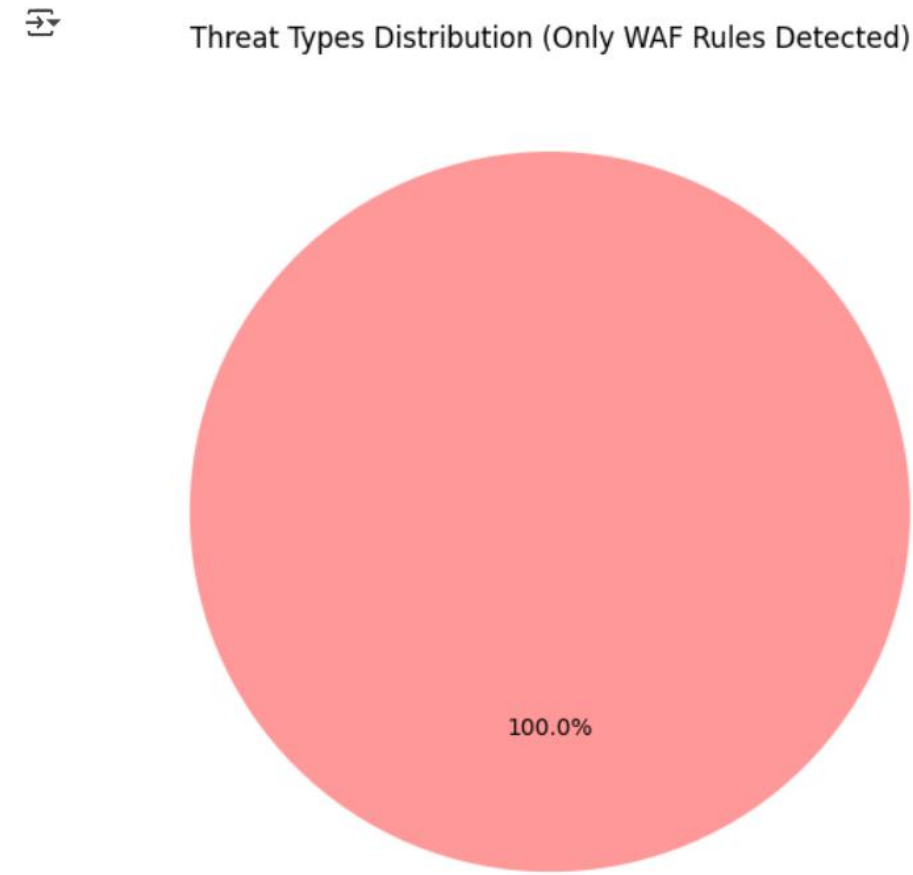
Touheed Khan
10:54 AM Today

Threat Type Distribution

The dataset currently contains only one type of detected threat: `waf_rule`.
Hence, the distribution is 100% under this category.

This could mean:
- The system was only monitoring WAF-based rules
- Or logs were filtered before export


Distribution of Threat Types
100.0%
waf_rule


Threat Types Distribution (Only WAF Rules Detected)
100.0%

```
[56] import networkx as nx
     import matplotlib.pyplot as plt
```

```
     import networkx as nx
     import matplotlib.pyplot as plt
```

```
[61] print(df.columns)
```

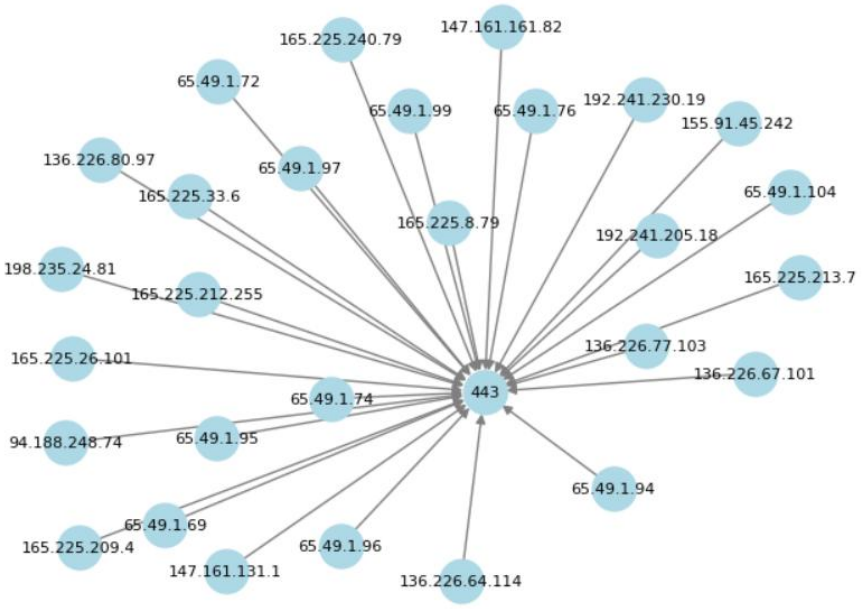
```
Index(['bytes_in', 'bytes_out', 'creation_time', 'end_time', 'src_ip',
       'src_ip_country_code', 'response.code', 'dst_port', 'observation_name',
       'source.meta', 'source.name', 'time', 'detection_types', 'Year',
       'Month', 'Day', 'Hour', 'Weekday', 'session_duration', 'only_date'],
      dtype='object')
```

```
[62] # Step 1: Clean up missing data
     df_clean = df[df['src_ip'].notna() & df['dst_port'].notna()]
```

```
[63] # Step 2: Convert dst_port to string (nodes must be string)
     df_clean['dst_port'] = df_clean['dst_port'].astype(str)
```

```
[64] # Step 3: Build the directed graph
     G = nx.from_pandas_edgelist(
         df_clean,
         source='src_ip',
         target='dst_port',
         create_using=nx.DiGraph()
     )
```

```
[75] plt.savefig("network_graph.png", dpi=300, bbox_inches='tight')
     plt.savefig("chart_name.png", dpi=300)


     # Step 4: Draw graph
     nx.draw(
         G,
         pos,
         with_labels=False,
         node_color='lightblue',
         edge_color='gray',
         node_size=500,
         arrows=True
     )
     labels = {node: str(node) for node in G.nodes()}
     nx.draw_networkx_labels(G, pos, labels, font_size=8, font_color='black')
```



Submission Links & Sources:

Final Report PDF:

Click to View PDF Report

Project Explanation Video:

Watch Video

Dataset Source:

Download Dataset