Final Project STAT PROB

Hands-on Exploratory Data Analysis (EDA)

EDA is the initial step in data analysis, aimed at understanding the characteristics, structure, and patterns within a dataset before applying statistical modeling or machine learning.

EDA is typically carried out through:

- **Descriptive summaries** → simple statistics such as mean, median, standard deviation, and frequency counts.
- **Visualization** → plots (histograms, scatter plots, boxplots, heatmaps, etc.) to uncover patterns, relationships between variables, and anomalies.

0. Persiapan

1. Python library that'll be used are:

- Pandas → for data manipulation and analysis
- NumPy → for numerical computing
- Matplotlib → for basic plotting and visualization
- Seaborn → for advanced statistical data visualization

2. Dataset is from:

Kaggle: https://www.kaggle.com/datasets/nasirayub2/drone-based-malware-detection-dbmd

In [2]:
```
#Mount file dari google
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:
```
%cd "/content/drive/MyDrive/STATPROB"
```

```
/content/drive/MyDrive/STATPROB
```

In [4]:
```
# import requirements library
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [4]:
```

```

In [5]:
```
# load dataset
data1 = pd.read_csv("/content/drive/MyDrive/STATPROB/DATASET/network_traffic_d
data2 = pd.read_csv("/content/drive/MyDrive/STATPROB/DATASET/malware_detectior
```

1. Deskripsi Dataset

- Jumlah observasi (baris) dan variabel (kolom)
- Tipe data setiap kolom (numerik, kategorikal, teks, tanggal, dll.)
- Informasi missing values
- Duplicate records
- Ringkasan statistik deskriptif (mean, median, min, max, std, dll.)

Dataset terdiri dari 2 file utama, row-aligned:

- network_traffic_data.csv
- malware_detection_data.csv

1.1 Network Traffic Data

In [6]:
```
# melihat sekilas data 5 row pertama
data1.head()
```

Out[6]:

	timestamp	source_ip	destination_ip	source_port	destination_port	pro
0	2024-06-28 19:02:55	86.230.134.129	3.46.98.34	3756	35357	
1	2022-12-09 13:36:47	37.211.177.132	40.109.190.27	44591	30823	
2	2022-08-22 04:59:38	54.129.79.47	77.250.43.217	43637	60156	
3	2023-02-07 15:30:41	157.59.116.46	211.120.32.204	53440	17944	
4	2020-12-08 21:06:49	161.125.154.101	62.15.8.89	46915	3390	

5 rows × 30 columns

In [7]:
```
# get the shape of the dataset
baris, kolom = data1.shape
print("baris:", baris)
print("kolom:", kolom)
```

```
baris: 200000
kolom: 30
```

In [8]:
```
# get info of the dataset
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 30 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   timestamp             200000 non-null  object 
 1   source_ip             200000 non-null  object 
 2   destination_ip        200000 non-null  object 
 3   source_port           200000 non-null  int64  
 4   destination_port      200000 non-null  int64  
 5   protocol              200000 non-null  object 
 6   packet_length         200000 non-null  int64  
 7   payload_data          200000 non-null  object 
 8   flag                  200000 non-null  object 
 9   traffic_volume        200000 non-null  float64
 10  flow_duration         200000 non-null  float64
 11  flow_bytes_per_s      200000 non-null  float64
 12  flow_packets_per_s    200000 non-null  float64
 13  packet_count          200000 non-null  int64  
 14  average_packet_size   200000 non-null  float64
 15  min_packet_size       200000 non-null  float64
 16  max_packet_size       200000 non-null  float64
 17  packet_size_variance  200000 non-null  float64
 18  header_length         200000 non-null  int64  
 19  payload_length        200000 non-null  int64  
 20  ip_ttl                200000 non-null  int64  
 21  tcp_window_size       200000 non-null  int64  
 22  icmp_type             200000 non-null  object 
 23  dns_query_count       200000 non-null  int64  
 24  dns_response_count    200000 non-null  int64  
 25  http_method           200000 non-null  object 
 26  http_status_code      200000 non-null  int64  
 27  content_type          200000 non-null  object 
 28  ssl_tls_version       200000 non-null  object 
 29  ssl_tls_cipher_suite  200000 non-null  object 
dtypes: float64(8), int64(11), object(11)
memory usage: 45.8+ MB
```

In [9]:

```
# get statistical summary
exclude_cols = ['source_port', 'destination_port','http_status_code','ip_ttl']
data1.drop(columns=exclude_cols).describe() # Run describe only on the remaini
```

Out[9]:

	packet_length	traffic_volume	flow_duration	flow_bytes_per_s	flow_pack
count	200000.000000	200000.000000	200000.000000	200000.000000	2000
mean	769.616195	500.114178	500.039276	500.076684	5
std	421.442038	289.085972	288.439345	288.590810	2
min	40.000000	0.001406	0.001422	0.022009	
25%	405.000000	249.778876	250.310140	250.572875	2
50%	769.000000	500.169651	499.945739	499.963350	5
75%	1135.000000	751.000848	748.884462	750.168943	7
max	1499.000000	999.998001	999.996836	999.995845	9

1.1.1a **Mengekstrak Variabel Numerik & Histogram**

1.1.1a.A **Traffic volume / rates**

- traffic_volume
- flow_duration
- flow_bytes_per_s
- average_packet_size
- min_packet_size
- max_packet_size
- packet_size_variance

In [10]:

```
# Columns from 1.1.1.c Traffic volume / rates
traffic_cols = ['traffic_volume','flow_duration','flow_bytes_per_s','average_p
# Convert all to numeric
# errors='coerce' -> any text/non-numeric will become NaN
data1[traffic_cols] = data1[traffic_cols].apply(pd.to_numeric, errors='coerce'
# Check dtypes to confirm numeric conversion
data1[traffic_cols].dtypes   # should show float64 or Int64
# 5Descriptive statistics for these variables
data1[traffic_cols].describe()
```

Out[10]:

	traffic_volume	flow_duration	flow_bytes_per_s	average_packet_size	mi
count	200000.000000	200000.000000	200000.000000	200000.000000	2
mean	500.114178	500.039276	500.076684	770.887896	
std	289.085972	288.439345	288.590810	421.525688	
min	0.001406	0.001422	0.022009	40.005137	
25%	249.778876	250.310140	250.572875	405.898694	
50%	500.169651	499.945739	499.963350	771.526962	
75%	751.000848	748.884462	750.168943	1136.386094	
max	999.998001	999.996836	999.995845	1499.987460	

1.1.1a.B **Other metrics**

- dns_query_count
- dns_response_count

1.1.1a.B.i **DNS Query Count**

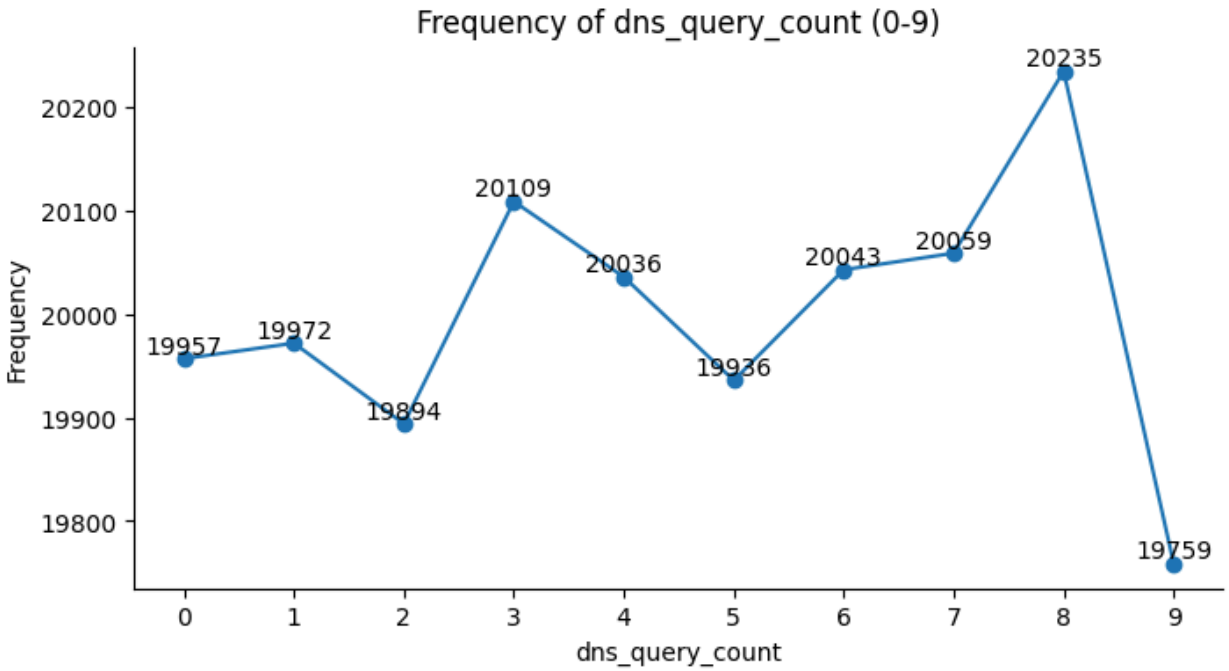
In [11]:

```
col = 'dns_query_count'
# Convert to numeric (if not already)
data1[col] = pd.to_numeric(data1[col], errors='coerce')
# Show frequency
print(f"\n🔹 Frequency of {col}")
freq_table = (
    data1[col]
    .value_counts(dropna=False)
    .sort_index()
    .to_frame(name='frequency')
)
display(freq_table)
```

```
🔹 Frequency of dns_query_count
```

dns_query_count	frequency
0	19957
1	19972
2	19894
3	20109
4	20036
5	19936
6	20043
7	20059
8	20235
9	19759

In [12]:
```
# Filter to only show values 0-9
filtered_freq = freq_table[(freq_table.index >= 0) & (freq_table.index <= 9)]

plt.figure(figsize=(8, 4))
plt.plot(filtered_freq.index, filtered_freq['frequency'], marker='o')
plt.title(f'Frequency of {col} (0-9)')
plt.xlabel(col)
plt.ylabel('Frequency')
plt.gca().spines[['top', 'right']].set_visible(False)

# FORCE x-axis to show every number 0–9
plt.xticks(filtered_freq.index)

# Add frequency labels
for x, y in zip(filtered_freq.index, filtered_freq['frequency']):
    plt.text(x, y, str(y), ha='center', va='bottom')

plt.show()
```



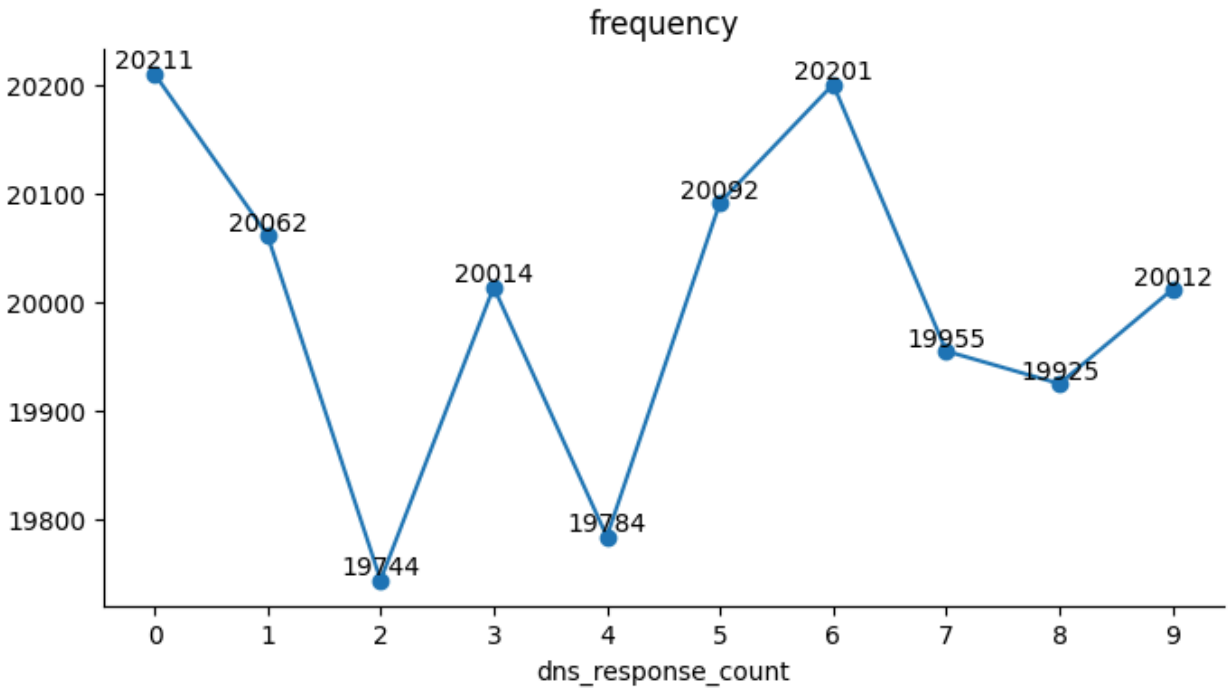
1.1.1a.B.ii **DNS Response Count**

In [65]:
```
col = 'dns_response_count'
# Convert to numeric (if not already)
data1[col] = pd.to_numeric(data1[col], errors='coerce')
# Show frequency
print(f"\n🔹 Frequency of {col}")
freq_table = (
    data1[col]
    .value_counts(dropna=False)
    .sort_index()
    .to_frame(name='frequency')
)
display(freq_table)
```

```
🔹 Frequency of dns_response_count
```

dns_response_count	frequency
0	20211
1	20062
2	19744
3	20014
4	19784
5	20092
6	20201
7	19955
8	19925
9	20012

In [66]:
```
# Filter dns_response_count to only show values 0-9
filtered_freq = freq_table[(freq_table.index >= 0) & (freq_table.index <= 9)]

# Plot the line graph with dots
ax = filtered_freq['frequency'].plot(kind='line', figsize=(8, 4), title='frequ
plt.gca().spines[['top', 'right']].set_visible(False)

# Force x-axis to show 0,1,2,...,9
plt.xticks(filtered_freq.index)

# Add frequency labels
for x, y in zip(filtered_freq.index, filtered_freq['frequency']):
    plt.text(x, y, str(y), ha='center', va='bottom')

plt.show()
```



1.1.1b **Mengekstrak Variabel Kategorikal & Histogram**

1.1.1b.A **Protocol**

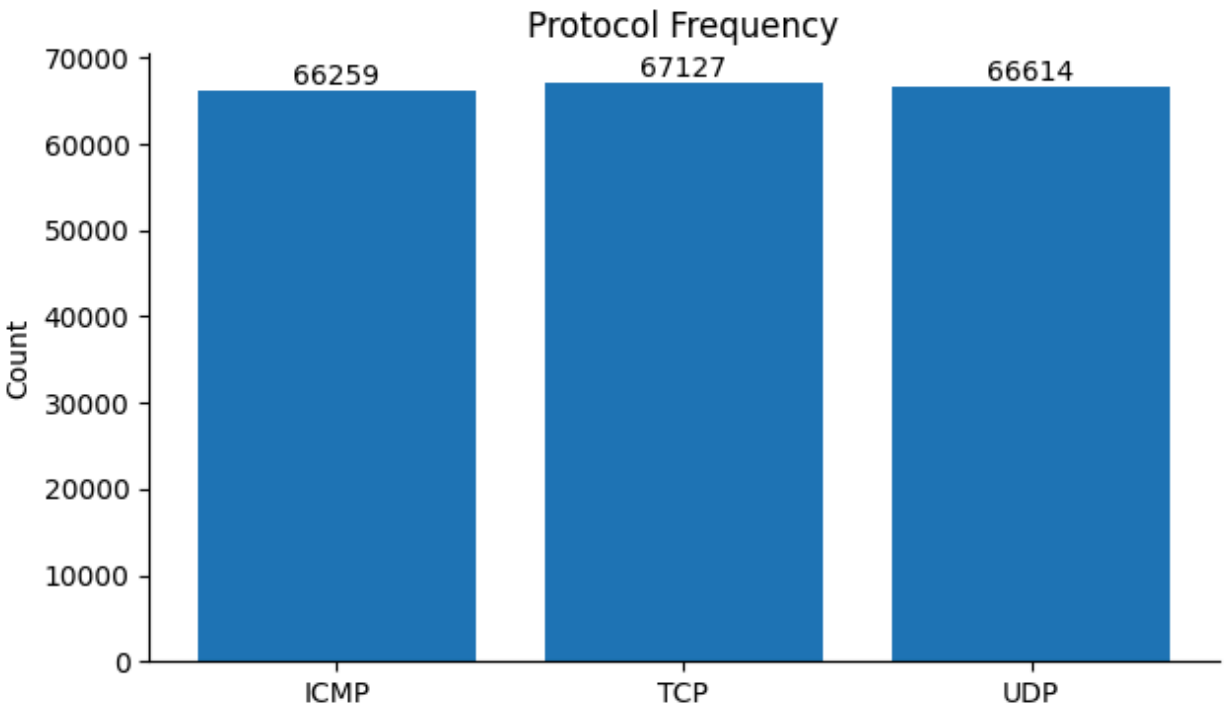
In [15]:
```
freq_table = (
    data1['protocol']
    .value_counts(dropna=False)    # count all categories, include NaN
    .sort_index()                  # sort alphabetically or numerically
    .to_frame(name='frequency')
)

# Just display it as a normal DataFrame
freq_table
```

Out[15]:

protocol	frequency
ICMP	66259
TCP	67127
UDP	66614

In [71]:
```
# Recalculate freq_table for 'protocol' to ensure correctness
freq_table = (
    data1['protocol']
    .value_counts(dropna=False)    # count all categories, include NaN
    .sort_index()                  # sort alphabetically or numerically
```

```
    .to_frame(name='frequency')
)

valid = ['ICMP','TCP','UDP']
df = freq_table[freq_table.index.isin(valid)]

df = df.loc[valid]     # forces the correct order

plt.figure(figsize=(7,4))
bars = plt.bar(df.index, df['frequency'])

plt.gca().spines[['top','right']].set_visible(False)
plt.title('Protocol Frequency')
plt.ylabel('Count')

for bar in bars:
    h = bar.get_height()
    plt.text(bar.get_x()+bar.get_width()/2, h, str(h),
             ha='center', va='bottom')

plt.show()
```



1.1.1b.B **Timestamp Kategori**

In [17]:
```
data1['timestamp'] = pd.to_datetime(data1['timestamp'], errors='coerce')
data1['year']  = data1['timestamp'].dt.year
data1['month'] = data1['timestamp'].dt.month
data1['day']   = data1['timestamp'].dt.day
data1['hour']  = data1['timestamp'].dt.hour
```

1.1.1b.B.i Frequency by Year

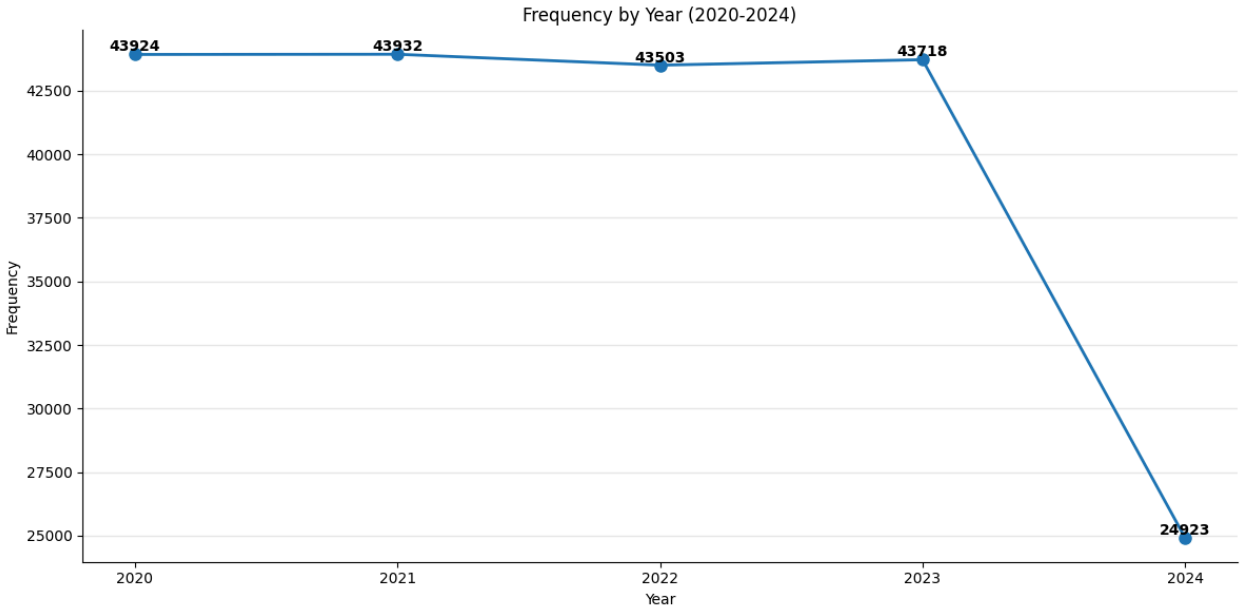
In [18]:
```
# Frequency by Year
print("Frequency by Year:")
year_freq = (
    data1['year']
    .value_counts(dropna=False)
    .sort_index()
    .to_frame(name='frequency')
)
display(year_freq)
```

Frequency by Year:

year	frequency
2020	43924
2021	43932
2022	43503
2023	43718
2024	24923

In [19]:
```
# Frequency by Year - filter only years 2020-2024
year_freq = (
    data1['year']
    .value_counts(dropna=False)
    .sort_index()
    .to_frame(name='frequency')
)

# Filter to only include years 2020-2024
filtered_years = [2020, 2021, 2022, 2023, 2024]
filtered_freq = year_freq[year_freq.index.isin(filtered_years)]

# Plot the line graph with dots
plt.figure(figsize=(12, 6))
plt.plot(filtered_freq.index, filtered_freq['frequency'], marker='o', linewidt
plt.title('Frequency by Year (2020-2024)')
plt.xlabel('Year')
plt.ylabel('Frequency')
plt.gca().spines[['top', 'right']].set_visible(False)

# Set x-axis ticks to only show the filtered years
plt.xticks(filtered_years)

# Add frequency values as annotations above the dots
for x, y in zip(filtered_freq.index, filtered_freq['frequency']):
    plt.text(x, y, str(y), ha='center', va='bottom', fontweight='bold')
```

```
plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
```



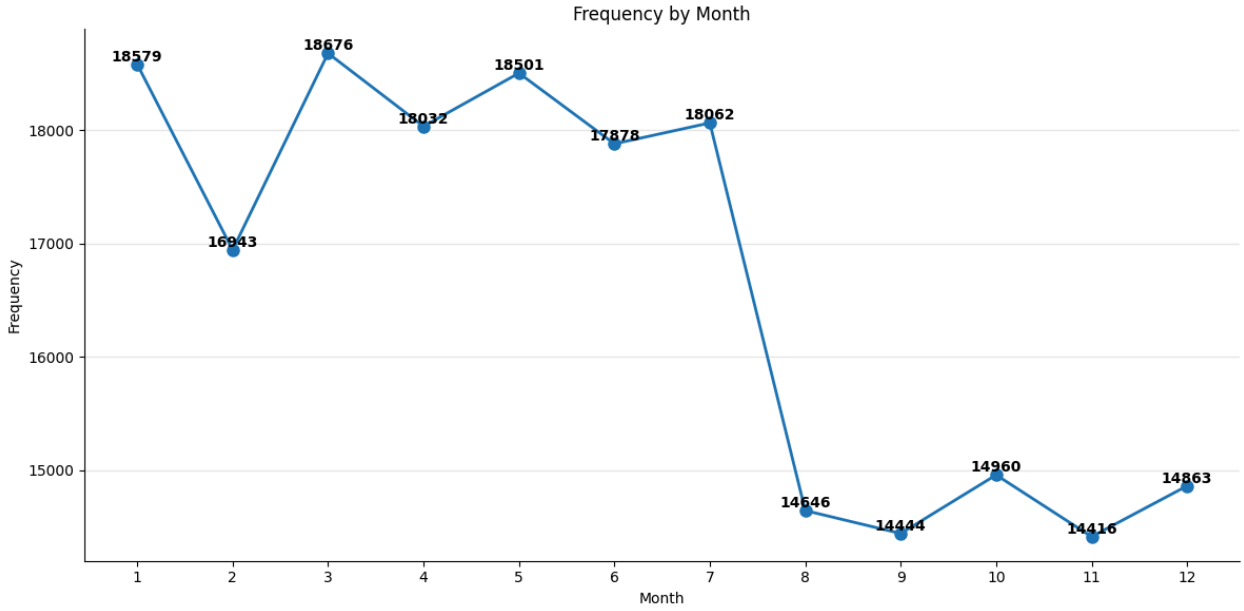
1.1.1b.B.ii Frequency by Month

In [20]:
```
# Frequency by Month
print("\nFrequency by Month:")
month_freq = (
    data1['month']
    .value_counts(dropna=False)
    .sort_index()
    .to_frame(name='frequency')
)
display(month_freq)
```

Frequency by Month:

month	frequency
1	18579
2	16943
3	18676
4	18032
5	18501
6	17878
7	18062
8	14646
9	14444
10	14960
11	14416
12	14863

In [21]:
```
# Plot the line graph with dots
plt.figure(figsize=(12, 6))
plt.plot(month_freq.index, month_freq['frequency'], marker='o', linewidth=2, m
plt.title('Frequency by Month')
plt.xlabel('Month')
plt.ylabel('Frequency')
plt.gca().spines[['top', 'right']].set_visible(False)

# Set x-axis to show all months (1-12)
plt.xticks(range(1, 13))

# Add frequency values as annotations above the dots
for x, y in zip(month_freq.index, month_freq['frequency']):
    plt.text(x, y, str(y), ha='center', va='bottom', fontweight='bold')

plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
```



1.1.1b.B.iii Frequency by Day

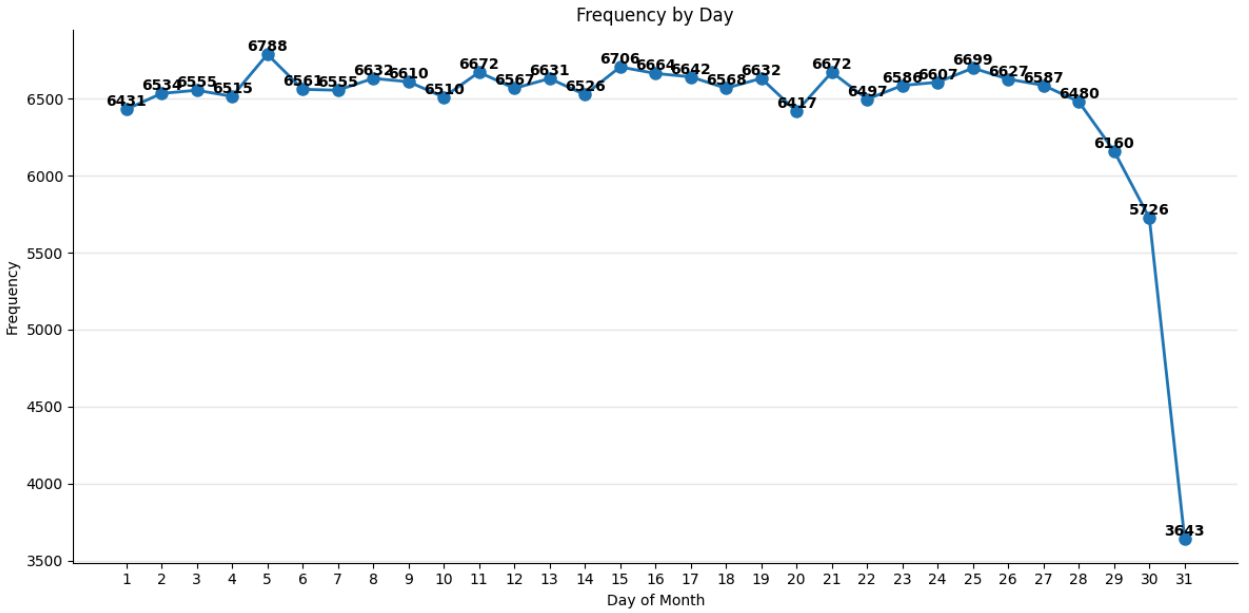
In [22]:
```
# Frequency by Day
print("\nFrequency by Day:")
day_freq = (
    data1['day']
    .value_counts(dropna=False)
    .sort_index()
    .to_frame(name='frequency')
)
display(day_freq)
```

```
Frequency by Day:
```

day	frequency
1	6431
2	6534
3	6555
4	6515
5	6788
6	6561
7	6555
8	6632
9	6610
10	6510
11	6672
12	6567
13	6631
14	6526
15	6706
16	6664
17	6642
18	6568
19	6632
20	6417
21	6672
22	6497
23	6586
24	6607
25	6699
26	6627
27	6587
28	6480
29	6160
30	5726

day	frequency
31	3643

In [23]:
```
# Plot the line graph with dots
plt.figure(figsize=(12, 6))
plt.plot(day_freq.index, day_freq['frequency'], marker='o', linewidth=2, marke
plt.title('Frequency by Day')
plt.xlabel('Day of Month')
plt.ylabel('Frequency')
plt.gca().spines[['top', 'right']].set_visible(False)

# Set x-axis to show all days (1-31)
plt.xticks(range(1, 32))

# Add frequency values as annotations above the dots
for x, y in zip(day_freq.index, day_freq['frequency']):
    plt.text(x, y, str(y), ha='center', va='bottom', fontweight='bold')

plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
```



1.1.1b.B.iv Frequency by Hour

In [24]:
```
# Frequency by Hour
print("\nFrequency by Hour:")
hour_freq = (
    data1['hour']
    .value_counts(dropna=False)
    .sort_index()
```

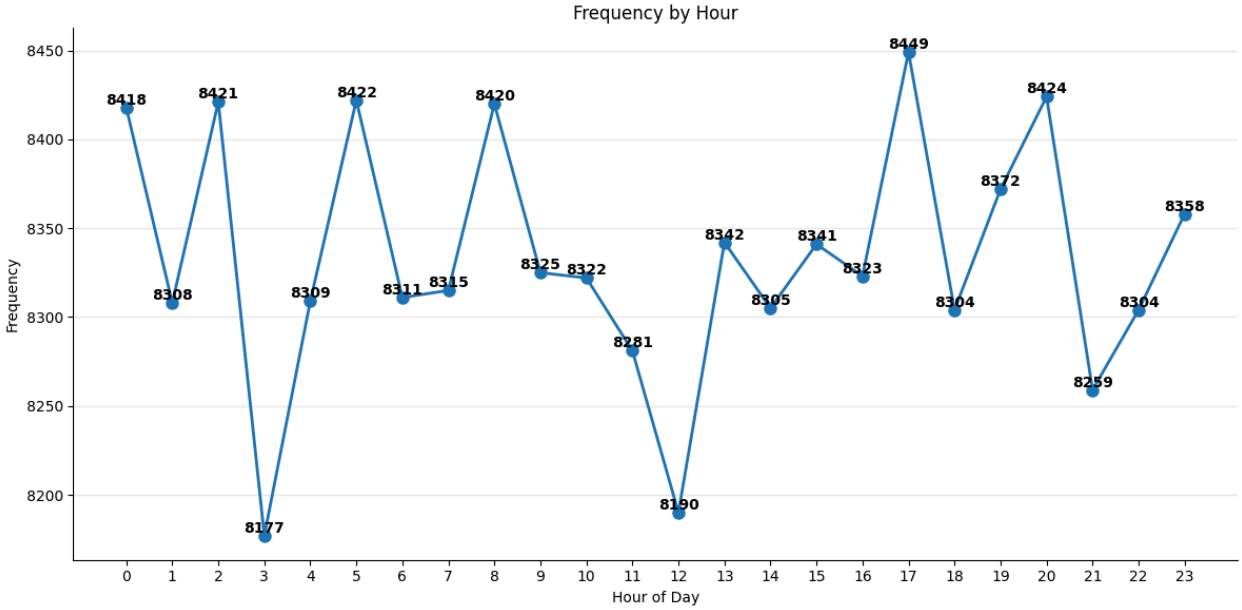
```
    .to_frame(name='frequency')
)
display(hour_freq)
```

```
Frequency by Hour:
```

hour	frequency
0	8418
1	8308
2	8421
3	8177
4	8309
5	8422
6	8311
7	8315
8	8420
9	8325
10	8322
11	8281
12	8190
13	8342
14	8305
15	8341
16	8323
17	8449
18	8304
19	8372
20	8424
21	8259
22	8304
23	8358

In [25]:
```
# Plot the line graph with dots
plt.figure(figsize=(12, 6))
plt.plot(hour_freq.index, hour_freq['frequency'], marker='o', linewidth=2, mar
plt.title('Frequency by Hour')
```

```
plt.xlabel('Hour of Day')
plt.ylabel('Frequency')
plt.gca().spines[['top', 'right']].set_visible(False)

# Set x-axis to show all hours (0-23)
plt.xticks(range(0, 24))

# Add frequency values as annotations above the dots
for x, y in zip(hour_freq.index, hour_freq['frequency']):
    plt.text(x, y, str(y), ha='center', va='bottom', fontweight='bold')

plt.grid(axis='y', alpha=0.3)
plt.tight_layout()
plt.show()
```



1.2 Malware Detection Data

In [26]:
```
# melihat sekilas data 5 row pertama
data2.head()
```

Out[26]:

	anomaly_score	suspicious_ip_count	malicious_payload_indicator	reputation_s
0	0.687331	5	1	82.67
1	0.519249	2	1	27.14
2	0.363944	7	0	21.24
3	0.890735	9	0	76.18
4	0.686994	8	0	12.93

In [27]:
```
# get the shape of the dataset
```

```
baris, kolom = data2.shape
print("baris:", baris)
print("kolom:", kolom)
```

```
baris: 200000
kolom: 10
```

In [28]:
```
# get info of the dataset
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 10 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   anomaly_score                200000 non-null  float64
 1   suspicious_ip_count          200000 non-null  int64  
 2   malicious_payload_indicator  200000 non-null  int64  
 3   reputation_score             200000 non-null  float64
 4   behavioral_score             200000 non-null  float64
 5   attack_type                  200000 non-null  object 
 6   signature_match              200000 non-null  int64  
 7   sandbox_result               200000 non-null  object 
 8   heuristic_score              200000 non-null  float64
 9   traffic_pattern              200000 non-null  object 
dtypes: float64(4), int64(3), object(3)
memory usage: 15.3+ MB
```

In [29]:
```
# get statistical summary
data2.describe()
```

Out[29]:

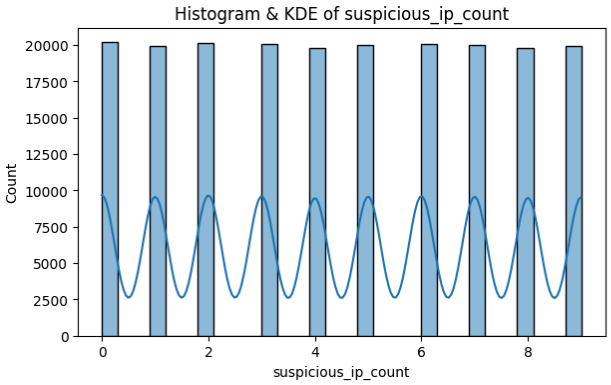
	anomaly_score	suspicious_ip_count	malicious_payload_indicator	reputat
count	200000.000000	200000.000000	200000.00000	2000
mean	0.500966	4.489850	0.49912	
std	0.288896	2.873703	0.50000	
min	0.000002	0.000000	0.00000	
25%	0.251379	2.000000	0.00000	
50%	0.500837	4.000000	0.00000	
75%	0.751856	7.000000	1.00000	
max	0.999994	9.000000	1.00000	

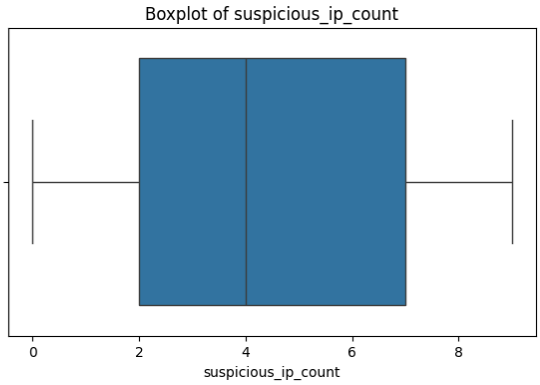
1.2.1 Mengekspor numerik

1.2.1.a suspicious_ip_count

In [30]:
```
plt.figure(figsize=(15,4))
```

```
plt.subplot(1,2,1)
sns.histplot(data2['suspicious_ip_count'], kde=True, bins=30)
plt.title("Histogram & KDE of suspicious_ip_count")

plt.subplot(1,2,2)
sns.boxplot(x=data2['suspicious_ip_count'])
plt.title("Boxplot of suspicious_ip_count")

plt.show()

# Tabel ringkasan vertikal
display(data2['suspicious_ip_count'].describe().to_frame(name="suspicious_ip_c
```





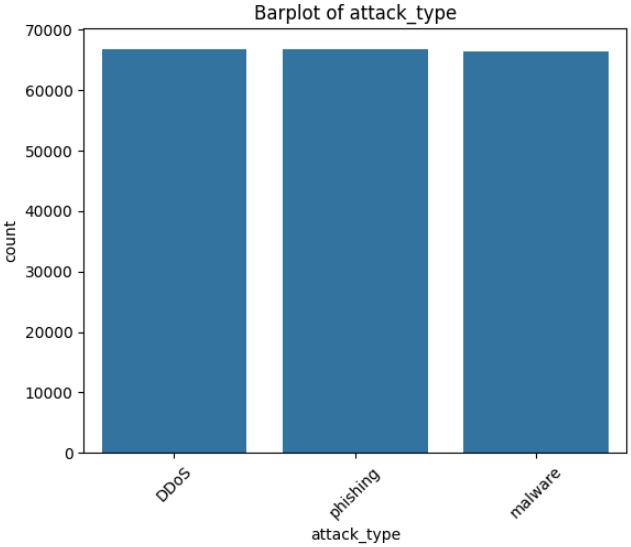
	suspicious_ip_count
count	200000.000000
mean	4.489850
std	2.873703
min	0.000000
25%	2.000000
50%	4.000000
75%	7.000000
max	9.000000

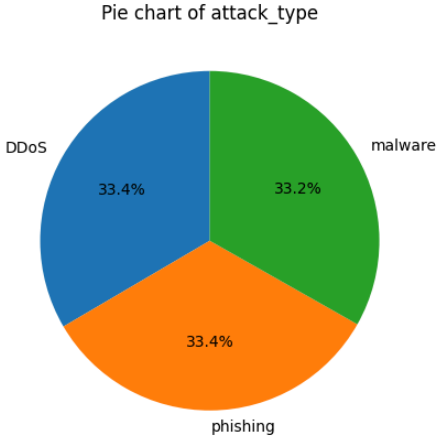
1.2.2 Variabel Kategorikal

1.2.2.a attack_type

In [31]:
```
plt.figure(figsize=(14,5))

plt.subplot(1,2,1)
sns.countplot(x=data2['attack_type'], order=data2['attack_type'].value_counts(
plt.title("Barplot of attack_type")
plt.xticks(rotation=45)
```

```
plt.subplot(1,2,2)
data2['attack_type'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90)
plt.title("Pie chart of attack_type")
plt.ylabel("")
plt.show()

# Tabel ringkasan vertikal
display(data2['attack_type'].value_counts().reset_index().rename(columns={'inc
```

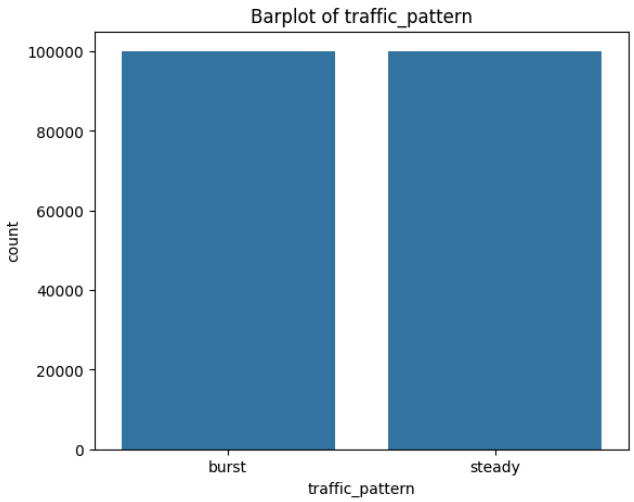


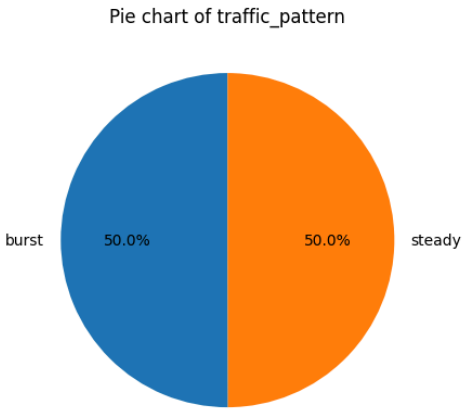


	jumlah	count
0	DDoS	66864
1	phishing	66751
2	malware	66385

1.2.2.c traffic_pattern

In [32]:
```
plt.figure(figsize=(14,5))

plt.subplot(1,2,1)
sns.countplot(x=data2['traffic_pattern'], order=data2['traffic_pattern'].value
plt.title("Barplot of traffic_pattern")

plt.subplot(1,2,2)
data2['traffic_pattern'].value_counts().plot.pie(autopct='%1.1f%%', startangle
plt.title("Pie chart of traffic_pattern")
plt.ylabel("")
plt.show()

display(data2['traffic_pattern'].value_counts().reset_index().rename(columns={
```





	jumlah	count
0	burst	100065
1	steady	99935

In [33]:
```
# get the shape of the dataset
baris, kolom = data2.shape
print("baris:", baris)
print("kolom:", kolom)
```

```
baris: 200000
kolom: 10
```

In [34]:
```
# get info of the dataset
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 10 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   anomaly_score                200000 non-null  float64
 1   suspicious_ip_count          200000 non-null  int64  
 2   malicious_payload_indicator  200000 non-null  int64  
 3   reputation_score             200000 non-null  float64
 4   behavioral_score             200000 non-null  float64
 5   attack_type                  200000 non-null  object 
 6   signature_match              200000 non-null  int64  
 7   sandbox_result               200000 non-null  object 
 8   heuristic_score              200000 non-null  float64
 9   traffic_pattern              200000 non-null  object 
dtypes: float64(4), int64(3), object(3)
memory usage: 15.3+ MB
```

In [35]:
```
# get statistical summary
data2.describe()
```

Out[35]:

	anomaly_score	suspicious_ip_count	malicious_payload_indicator	reputat
count	200000.000000	200000.000000	200000.00000	2000
mean	0.500966	4.489850	0.49912	
std	0.288896	2.873703	0.50000	
min	0.000002	0.000000	0.00000	
25%	0.251379	2.000000	0.00000	
50%	0.500837	4.000000	0.00000	
75%	0.751856	7.000000	1.00000	
max	0.999994	9.000000	1.00000	

2. Analisis Missing Values & Outlier

- Persentase data yang hilang per kolom
- Visualisasi missing values (heatmap, bar chart)
- Penanganan missing values (drop, imputasi, dsb.)
- Identifikasi outlier (boxplot, Z-score, IQR)
- Strategi penanganan outlier

PRESENTASE DATA YANG HILANG PER-KOLOM

In [36]:
```
# ====================================
# KODE TAMBAHAN: MENCARI DATA YANG HILANG PER KOLOM
# ====================================

# MENCARI DATA YANG HILANG PER KOLOM
print("=== MISSING VALUES PER KOLOM UNTUK DATA1 (network_traffic_data.csv) ===
missing_data1 = data1.isnull().sum()
print(missing_data1)
print(f"\nTotal missing values di data1: {missing_data1.sum()}")
print(f"Persentase missing values per kolom:\n{(missing_data1 / len(data1) * 1

print("\n=== MISSING VALUES PER KOLOM UNTUK DATA2 (malware_detection_data.csv)
missing_data2 = data2.isnull().sum()
print(missing_data2)
print(f"\nTotal missing values di data2: {missing_data2.sum()}")
print(f"Persentase missing values per kolom:\n{(missing_data2 / len(data2) * 1

# VISUALISASI SEBAGAI TABEL BERWARNA (menggunakan pandas styling untuk HTML di
# Buat DataFrame untuk tabel data1
missing_df1 = pd.DataFrame({
    'Kolom': missing_data1.index,
    'Jumlah Missing': missing_data1.values,
```

```
    'Persentase (%)': (missing_data1 / len(data1) * 100).round(2).values
})

# Buat DataFrame untuk tabel data2
missing_df2 = pd.DataFrame({
    'Kolom': missing_data2.index,
    'Jumlah Missing': missing_data2.values,
    'Persentase (%)': (missing_data2 / len(data2) * 100).round(2).values
})

# Fungsi untuk styling tabel dengan warna
def style_missing_table(df):
    # Gradasi warna pada kolom 'Persentase (%)' (merah untuk tinggi, hijau unt
    styled = df.style.background_gradient(subset=['Persentase (%)'], cmap='RdY

    # Highlight baris dengan persentase > 5% (background merah muda)
    def highlight_high_missing(val):
        if isinstance(val, (int, float)) and val > 5:
            return 'background-color: #ffcccc'  # Merah muda untuk persentase
        return ''

    # Highlight baris dengan jumlah missing > 0 (background kuning muda)
    def highlight_any_missing(val):
        if isinstance(val, int) and val > 0:
            return 'background-color: #fff2cc'  # Kuning muda untuk ada missin
        return ''

    # Terapkan conditional formatting
    styled = styled.applymap(highlight_high_missing, subset=['Persentase (%)']
    styled = styled.applymap(highlight_any_missing, subset=['Jumlah Missing'])

    # Format angka
    styled = styled.format({
        'Jumlah Missing': '{:,.0f}',
        'Persentase (%)': '{:.2f}%'
    })

    # Tambahkan border dan font untuk keterbacaan
    styled = styled.set_table_styles([
        {'selector': 'th', 'props': [('font-weight', 'bold'), ('text-align', '
        {'selector': 'td', 'props': [('text-align', 'center'), ('padding', '8p
    ])

    return styled

# Tampilkan tabel berwarna untuk data1
print("\n=== TABEL VISUALISASI MISSING VALUES UNTUK DATA1 (network_traffic_dat
display(missing_df1.pipe(style_missing_table))

# Tampilkan tabel berwarna untuk data2
print("\n=== TABEL VISUALISASI MISSING VALUES UNTUK DATA2 (malware_detection_d
display(missing_df2.pipe(style_missing_table))
```

```python
# Ringkasan total (opsional, tetap teks)
print(f"\nTotal missing values di data1: {missing_data1.sum()}")
print(f"Total missing values di data2: {missing_data2.sum()}")
```

```
=== MISSING VALUES PER KOLOM UNTUK DATA1 (network_traffic_data.csv) ===
timestamp               0
source_ip               0
destination_ip          0
source_port             0
destination_port        0
protocol                0
packet_length           0
payload_data            0
flag                    0
traffic_volume          0
flow_duration           0
flow_bytes_per_s        0
flow_packets_per_s      0
packet_count            0
average_packet_size     0
min_packet_size         0
max_packet_size         0
packet_size_variance    0
header_length           0
payload_length          0
ip_ttl                  0
tcp_window_size         0
icmp_type               0
dns_query_count         0
dns_response_count      0
http_method             0
http_status_code        0
content_type            0
ssl_tls_version         0
ssl_tls_cipher_suite    0
year                    0
month                   0
day                     0
hour                    0
dtype: int64

Total missing values di data1: 0
Persentase missing values per kolom:
timestamp               0.0
source_ip               0.0
destination_ip          0.0
source_port             0.0
destination_port        0.0
protocol                0.0
packet_length           0.0
payload_data            0.0
flag                    0.0
traffic_volume          0.0
flow_duration           0.0
flow_bytes_per_s        0.0
flow_packets_per_s      0.0
packet_count            0.0
average_packet_size     0.0
```

```
min_packet_size         0.0
max_packet_size         0.0
packet_size_variance    0.0
header_length           0.0
payload_length          0.0
ip_ttl                  0.0
tcp_window_size         0.0
icmp_type               0.0
dns_query_count         0.0
dns_response_count      0.0
http_method             0.0
http_status_code        0.0
content_type            0.0
ssl_tls_version         0.0
ssl_tls_cipher_suite    0.0
year                    0.0
month                   0.0
day                     0.0
hour                    0.0
dtype: float64

=== MISSING VALUES PER KOLOM UNTUK DATA2 (malware_detection_data.csv) ===
anomaly_score                 0
suspicious_ip_count           0
malicious_payload_indicator   0
reputation_score              0
behavioral_score              0
attack_type                   0
signature_match               0
sandbox_result                0
heuristic_score               0
traffic_pattern               0
dtype: int64

Total missing values di data2: 0
Persentase missing values per kolom:
anomaly_score                 0.0
suspicious_ip_count           0.0
malicious_payload_indicator   0.0
reputation_score              0.0
behavioral_score              0.0
attack_type                   0.0
signature_match               0.0
sandbox_result                0.0
heuristic_score               0.0
traffic_pattern               0.0
dtype: float64

=== TABEL VISUALISASI MISSING VALUES UNTUK DATA1 (network_traffic_data.csv) ===
```

```
/tmp/ipython-input-2329948180.py:51: FutureWarning: Styler.applymap has been de
precated. Use Styler.map instead.
  styled = styled.applymap(highlight_high_missing, subset=['Persentase (%)'])
/tmp/ipython-input-2329948180.py:52: FutureWarning: Styler.applymap has been de
precated. Use Styler.map instead.
  styled = styled.applymap(highlight_any_missing, subset=['Jumlah Missing'])
```

	Kolom	Jumlah Missing	Persentase (%)
0	timestamp	0	0.00%
1	source_ip	0	0.00%
2	destination_ip	0	0.00%
3	source_port	0	0.00%
4	destination_port	0	0.00%
5	protocol	0	0.00%
6	packet_length	0	0.00%
7	payload_data	0	0.00%
8	flag	0	0.00%
9	traffic_volume	0	0.00%
10	flow_duration	0	0.00%
11	flow_bytes_per_s	0	0.00%
12	flow_packets_per_s	0	0.00%
13	packet_count	0	0.00%
14	average_packet_size	0	0.00%
15	min_packet_size	0	0.00%
16	max_packet_size	0	0.00%
17	packet_size_variance	0	0.00%
18	header_length	0	0.00%
19	payload_length	0	0.00%
20	ip_ttl	0	0.00%
21	tcp_window_size	0	0.00%
22	icmp_type	0	0.00%
23	dns_query_count	0	0.00%
24	dns_response_count	0	0.00%
25	http_method	0	0.00%
26	http_status_code	0	0.00%
27	content_type	0	0.00%

	Kolom	Jumlah Missing	Persentase (%)
28	ssl_tls_version	0	0.00%
29	ssl_tls_cipher_suite	0	0.00%
30	year	0	0.00%
31	month	0	0.00%
32	day	0	0.00%
33	hour	0	0.00%

```
=== TABEL VISUALISASI MISSING VALUES UNTUK DATA2 (malware_detection_data.csv)
===
/tmp/ipython-input-2329948180.py:51: FutureWarning: Styler.applymap has been de
precated. Use Styler.map instead.
  styled = styled.applymap(highlight_high_missing, subset=['Persentase (%)'])
/tmp/ipython-input-2329948180.py:52: FutureWarning: Styler.applymap has been de
precated. Use Styler.map instead.
  styled = styled.applymap(highlight_any_missing, subset=['Jumlah Missing'])
```

	Kolom	Jumlah Missing	Persentase (%)
0	anomaly_score	0	0.00%
1	suspicious_ip_count	0	0.00%
2	malicious_payload_indicator	0	0.00%
3	reputation_score	0	0.00%
4	behavioral_score	0	0.00%
5	attack_type	0	0.00%
6	signature_match	0	0.00%
7	sandbox_result	0	0.00%
8	heuristic_score	0	0.00%
9	traffic_pattern	0	0.00%

```
Total missing values di data1: 0
Total missing values di data2: 0
```

In [37]:
```
print("Menampilkan 5 baris pertama dari data1:")
display(data1.head())

print("\nMenampilkan 5 baris pertama dari data2:")
display(data2.head())
```

```
Menampilkan 5 baris pertama dari data1:
```

	timestamp	source_ip	destination_ip	source_port	destination_port	prot
0	2024-06-28 19:02:55	86.230.134.129	3.46.98.34	3756	35357	I
1	2022-12-09 13:36:47	37.211.177.132	40.109.190.27	44591	30823	I
2	2022-08-22 04:59:38	54.129.79.47	77.250.43.217	43637	60156	
3	2023-02-07 15:30:41	157.59.116.46	211.120.32.204	53440	17944	
4	2020-12-08 21:06:49	161.125.154.101	62.15.8.89	46915	3390	I

5 rows × 34 columns

Menampilkan 5 baris pertama dari data2:

	anomaly_score	suspicious_ip_count	malicious_payload_indicator	reputation_sc
0	0.687331	5	1	82.673
1	0.519249	2	1	27.140
2	0.363944	7	0	21.243
3	0.890735	9	0	76.188
4	0.686994	8	0	12.935

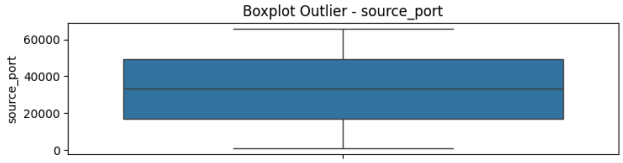
Identifikasi outlier (boxplot, Z-score, IQR)

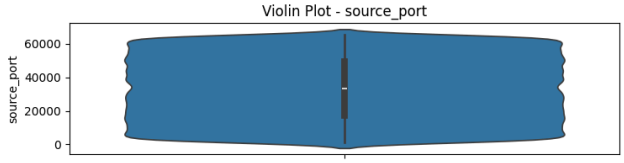
In [38]:
```
# Fungsi untuk mendeteksi outlier menggunakan Z-score (threshold = 3)
import scipy.stats as stats # Import the stats module
def detect_outliers_zscore(df, col, threshold=3):
    z_scores = np.abs(stats.zscore(df[col].dropna()))
    outliers = df[col][z_scores > threshold]
    return len(outliers), outliers.index.tolist()

# Fungsi untuk mendeteksi outlier menggunakan IQR
def detect_outliers_iqr(df, col, multiplier=1.5):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - multiplier * IQR
    upper_bound = Q3 + multiplier * IQR
    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)][col]
    return len(outliers), outliers.index.tolist()

# Identifikasi kolom numerik (exclude non-numeric seperti 'label' jika ada)
numeric_cols_data1 = data1.select_dtypes(include=[np.number]).columns.tolist()
numeric_cols_data2 = data2.select_dtypes(include=[np.number]).columns.tolist()
```

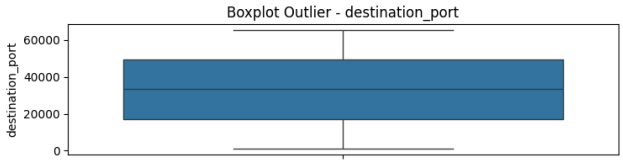
```
print("=== KOLOM NUMERIK UNTUK ANALISIS OUTLIER ===")
print("Data1:", numeric_cols_data1)
print("Data2:", numeric_cols_data2)

# ANALISIS DATA 1

print("\n=== IDENTIFIKASI OUTLIER UNTUK DATA1 (network_traffic_data.csv) ===")
# 1. BOXPLOT DAN VIOLIN PLOT UNTUK VISUALISASI OUTLIER (Data1)
plt.figure(figsize=(15, len(numeric_cols_data1) * 2)) # Adjust figure height b
for i, col in enumerate(numeric_cols_data1, 1):
    plt.subplot(len(numeric_cols_data1), 2, (i*2)-1)
    sns.boxplot(y=data1[col].dropna())
    plt.title(f'Boxplot Outlier - {col}')
    plt.subplot(len(numeric_cols_data1), 2, i*2)
    sns.violinplot(y=data1[col].dropna())
    plt.title(f'Violin Plot - {col}')
plt.tight_layout()
plt.show()


# 2. Z-SCORE DAN IQR UNTUK DATA1
outliers_zscore_data1 = {}
outliers_iqr_data1 = {}
for col in numeric_cols_data1:
    if data1[col].dropna().std() > 0:  # Hindari kolom dengan std=0
        num_z, idx_z = detect_outliers_zscore(data1, col)
        num_iqr, idx_iqr = detect_outliers_iqr(data1, col)
        outliers_zscore_data1[col] = {'jumlah': num_z, 'indeks': idx_z}
        outliers_iqr_data1[col] = {'jumlah': num_iqr, 'indeks': idx_iqr}
        print(f"\nKolom '{col}':")
        print(f"  Z-score Outliers: {num_z} (indeks: {idx_z[:5]}...)")  # Tamp
        print(f"  IQR Outliers: {num_iqr} (indeks: {idx_iqr[:5]}...)")

# Ringkasan total outlier
total_z_data1 = sum(d['jumlah'] for d in outliers_zscore_data1.values())
total_iqr_data1 = sum(d['jumlah'] for d in outliers_iqr_data1.values())
print(f"\nTotal Outlier Z-score di Data1: {total_z_data1}")
print(f"Total Outlier IQR di Data1: {total_iqr_data1}")

# ANALISIS UNTUK DATA 2
print("\n=== IDENTIFIKASI OUTLIER UNTUK DATA2 (malware_detection_data.csv) ===

# 1. BOXPLOT DAN VIOLIN PLOT UNTUK VISUALISASI OUTLIER (Data2)
plt.figure(figsize=(15, len(numeric_cols_data2) * 2)) # Adjust figure height b
for i, col in enumerate(numeric_cols_data2, 1):
    plt.subplot(len(numeric_cols_data2), 2, (i*2)-1)
    sns.boxplot(y=data2[col].dropna())
    plt.title(f'Boxplot Outlier - {col}')
    plt.subplot(len(numeric_cols_data2), 2, i*2)
    sns.violinplot(y=data2[col].dropna())
    plt.title(f'Violin Plot - {col}')
plt.tight_layout()
plt.show()
```

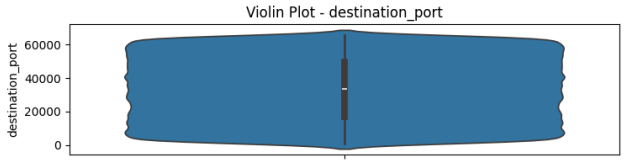
```python
# 2. Z-SCORE DAN IQR UNTUK DATA2
outliers_zscore_data2 = {}
outliers_iqr_data2 = {}
for col in numeric_cols_data2:
    if data2[col].dropna().std() > 0:  # Hindari kolom dengan std=0
        num_z, idx_z = detect_outliers_zscore(data2, col)
        num_iqr, idx_iqr = detect_outliers_iqr(data2, col)
        outliers_zscore_data2[col] = {'jumlah': num_z, 'indeks': idx_z}
        outliers_iqr_data2[col] = {'jumlah': num_iqr, 'indeks': idx_iqr}
        print(f"\nKolom '{col}':")
        print(f"  Z-score Outliers: {num_z} (indeks: {idx_z[:5]}...)")  # Tamp
        print(f"  IQR Outliers: {num_iqr} (indeks: {idx_iqr[:5]}...)")

# Ringkasan total outlier
total_z_data2 = sum(d['jumlah'] for d in outliers_zscore_data2.values())
total_iqr_data2 = sum(d['jumlah'] for d in outliers_iqr_data2.values())
print(f"\nTotal Outlier Z-score di Data2: {total_z_data2}")
print(f"Total Outlier IQR di Data2: {total_iqr_data2}")
```

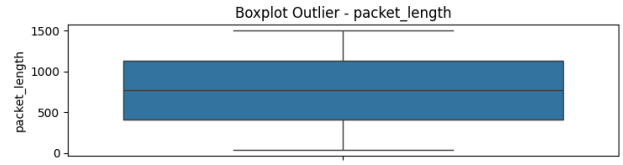
```
=== KOLOM NUMERIK UNTUK ANALISIS OUTLIER ===
Data1: ['source_port', 'destination_port', 'packet_length', 'traffic_volume',
'flow_duration', 'flow_bytes_per_s', 'flow_packets_per_s', 'packet_count', 'ave
rage_packet_size', 'min_packet_size', 'max_packet_size', 'packet_size_varianc
e', 'header_length', 'payload_length', 'ip_ttl', 'tcp_window_size', 'dns_quer
y_count', 'dns_response_count', 'http_status_code', 'year', 'month', 'day', 'ho
ur']
Data2: ['anomaly_score', 'suspicious_ip_count', 'malicious_payload_indicator',
'reputation_score', 'behavioral_score', 'signature_match', 'heuristic_score']

=== IDENTIFIKASI OUTLIER UNTUK DATA1 (network_traffic_data.csv) ===
```

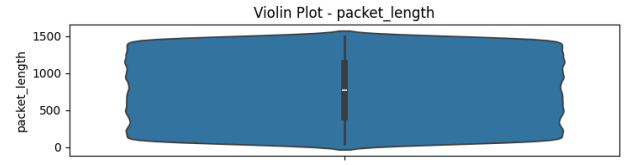

Boxplot Outlier - source_port
source_port
60000
40000
20000
0

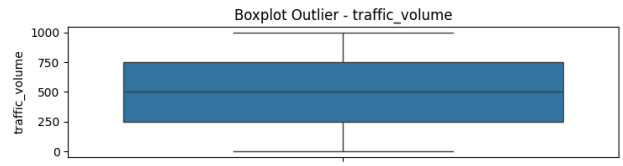

Violin Plot - source_port
source_port
60000
40000
20000
0

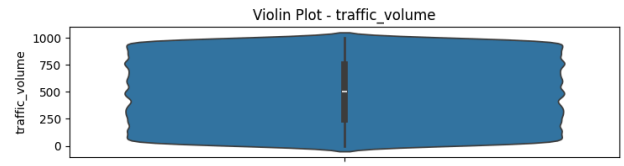

Boxplot Outlier - destination_port
destination_port
60000
40000
20000
0

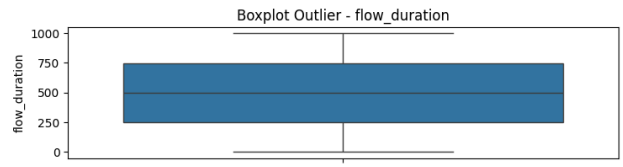

Violin Plot - destination_port
destination_port
60000
40000
20000
0

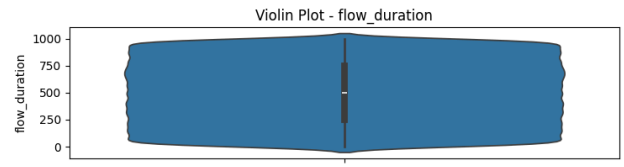

Boxplot Outlier - packet_length
packet_length
1500
1000
500
0

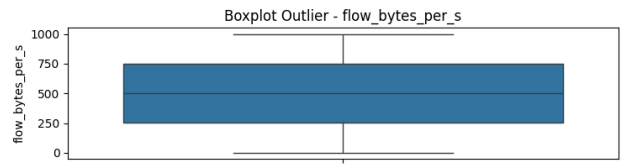

Violin Plot - packet_length
packet_length
1500
1000
500
0

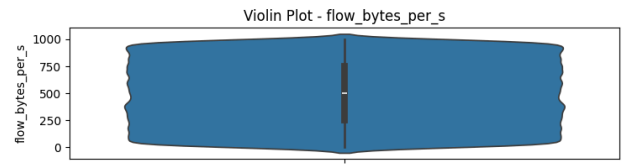

Boxplot Outlier - traffic_volume
traffic_volume
1000
750
500
250
0

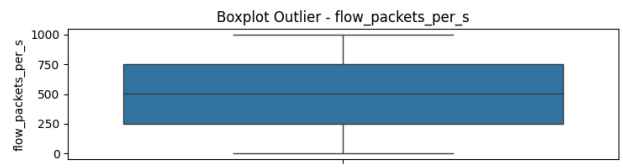

Violin Plot - traffic_volume
traffic_volume
1000
750
500
250
0

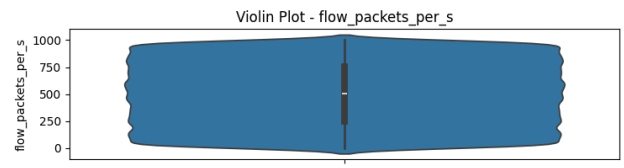

Boxplot Outlier - flow_duration
flow_duration
1000
750
500
250
0

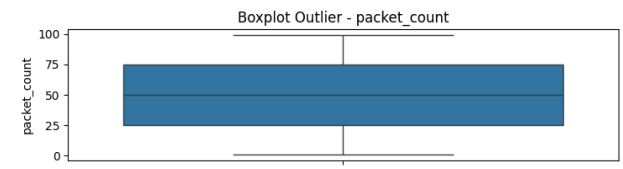

Violin Plot - flow_duration
flow_duration
1000
750
500
250
0

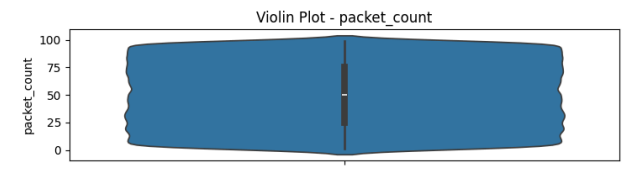

Boxplot Outlier - flow_bytes_per_s
flow_bytes_per_s
1000
750
500
250
0

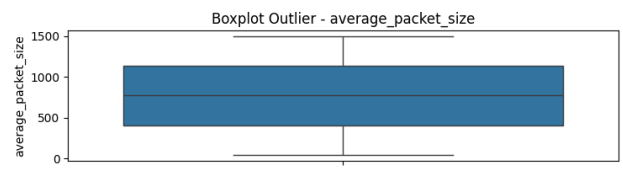

Violin Plot - flow_bytes_per_s
flow_bytes_per_s
1000
750
500
250
0

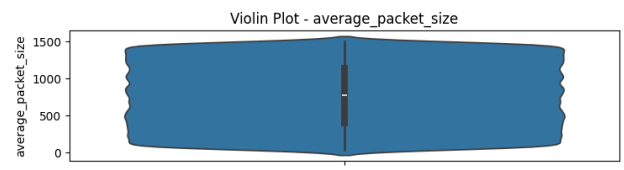

Boxplot Outlier - flow_packets_per_s
flow_packets_per_s
1000
750
500
250
0

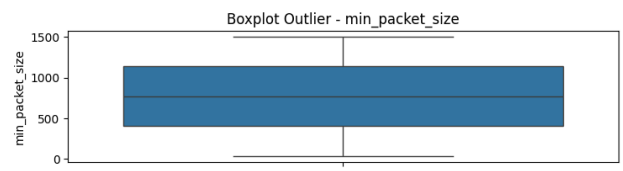

Violin Plot - flow_packets_per_s
flow_packets_per_s
1000
750
500
250
0

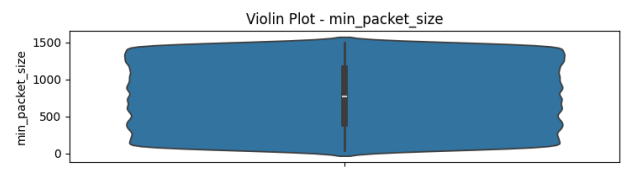

Boxplot Outlier - packet_count
packet_count
100
75
50
25
0

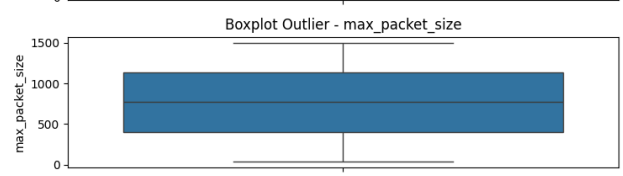

Violin Plot - packet_count
packet_count
100
75
50
25
0

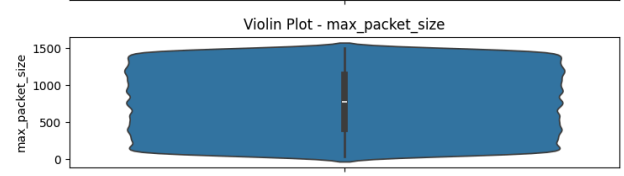

Boxplot Outlier - average_packet_size
average_packet_size
1500
1000
500
0


Violin Plot - average_packet_size
average_packet_size
1500
1000
500
0


Boxplot Outlier - min_packet_size
min_packet_size
1500
1000
500
0

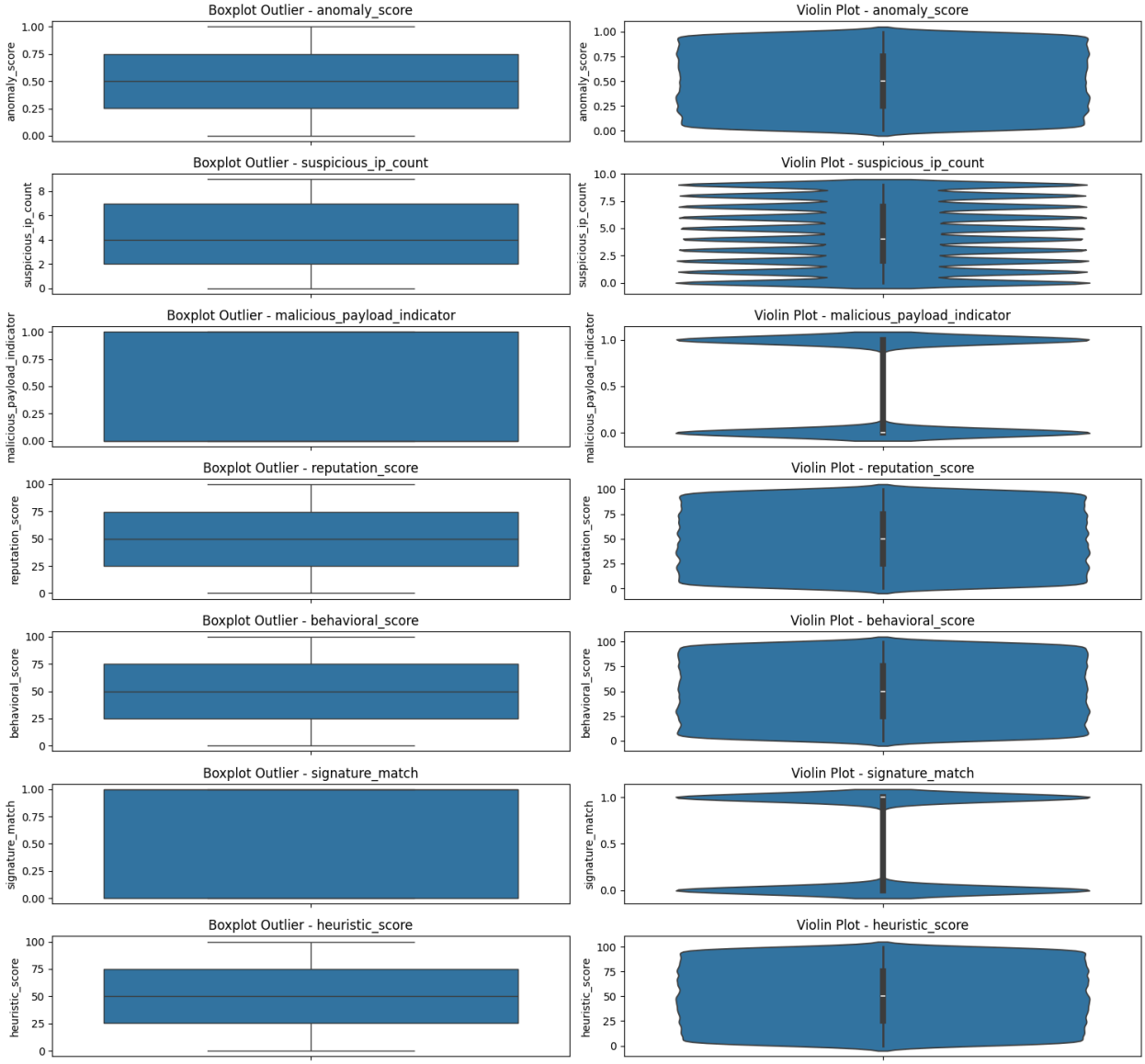

Violin Plot - min_packet_size
min_packet_size
1500
1000
500
0


Boxplot Outlier - max_packet_size
max_packet_size
1500
1000
500
0


Violin Plot - max_packet_size
max_packet_size
1500
1000
500
0

```
Kolom 'source_port':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'destination_port':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'packet_length':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'traffic_volume':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'flow_duration':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'flow_bytes_per_s':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'flow_packets_per_s':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'packet_count':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'average_packet_size':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'min_packet_size':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'max_packet_size':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'packet_size_variance':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'header_length':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'payload_length':
  Z-score Outliers: 0 (indeks: []...)
```

```
  IQR Outliers: 0 (indeks: []...)

Kolom 'ip_ttl':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'tcp_window_size':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'dns_query_count':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'dns_response_count':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'http_status_code':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'year':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'month':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'day':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'hour':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Total Outlier Z-score di Data1: 0
Total Outlier IQR di Data1: 0

=== IDENTIFIKASI OUTLIER UNTUK DATA2 (malware_detection_data.csv) ===
```


Boxplot Outlier - anomaly_score
anomaly_score
Violin Plot - anomaly_score
anomaly_score
Boxplot Outlier - suspicious_ip_count
suspicious_ip_count
Violin Plot - suspicious_ip_count
suspicious_ip_count
Boxplot Outlier - malicious_payload_indicator
malicious_payload_indicator
Violin Plot - malicious_payload_indicator
malicious_payload_indicator
Boxplot Outlier - reputation_score
reputation_score
Violin Plot - reputation_score
reputation_score
Boxplot Outlier - behavioral_score
behavioral_score
Violin Plot - behavioral_score
behavioral_score
Boxplot Outlier - signature_match
signature_match
Violin Plot - signature_match
signature_match
Boxplot Outlier - heuristic_score
heuristic_score
Violin Plot - heuristic_score
heuristic_score

```
Kolom 'anomaly_score':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'suspicious_ip_count':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'malicious_payload_indicator':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'reputation_score':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'behavioral_score':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'signature_match':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Kolom 'heuristic_score':
  Z-score Outliers: 0 (indeks: []...)
  IQR Outliers: 0 (indeks: []...)

Total Outlier Z-score di Data2: 0
Total Outlier IQR di Data2: 0
```

Strategi penanganan outlier

In [39]:

```python
# --- Bagian Penanganan Outlier ---
# Membuat salinan dataset untuk menerapkan strategi penanganan outlier tanpa m
data1_handled_delete = data1.copy()
data1_handled_cap = data1.copy()
data1_handled_median = data1.copy()
data2_handled_delete = data2.copy()
data2_handled_cap = data2.copy()
data2_handled_median = data2.copy()

# Fungsi untuk menghitung batas bawah dan atas menggunakan metode IQR
def get_iqr_bounds(df, col, multiplier=1.5):
    """Menghitung batas outlier menggunakan metode IQR."""
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - multiplier * IQR
    upper_bound = Q3 + multiplier * IQR
    return lower_bound, upper_bound

# Fungsi untuk menghapus baris yang mengandung outlier pada kolom tertentu
```

```python
def remove_outliers(df, col, multiplier=1.5):
    """Menghapus baris dengan outlier berdasarkan metode IQR."""
    lower, upper = get_iqr_bounds(df, col, multiplier)
    mask = (df[col] >= lower) & (df[col] <= upper)
    return df[mask]

# Fungsi untuk mengganti outlier dengan nilai batas (capping)
def cap_outliers(df, col, multiplier=1.5):
    """Melakukan capping (membatasi) outlier berdasarkan metode IQR."""
    lower, upper = get_iqr_bounds(df, col, multiplier)
    df_cap = df.copy()
    # Mengganti nilai di bawah batas bawah dengan batas bawah
    df_cap.loc[df_cap[col] < lower, col] = lower
    # Mengganti nilai di atas batas atas dengan batas atas
    df_cap.loc[df_cap[col] > upper, col] = upper
    return df_cap

# Fungsi untuk mengisi outlier dengan nilai median kolom
def replace_outliers_with_median(df, col, multiplier=1.5):
    """Mengganti outlier dengan nilai median kolom."""
    lower, upper = get_iqr_bounds(df, col, multiplier)
    median_val = df[col].median()
    df_med = df.copy()
    # Mengganti nilai outlier (baik di bawah batas bawah atau di atas batas at
    df_med.loc[(df_med[col] < lower) | (df_med[col] > upper), col] = median_va
    return df_med

# Identifikasi kolom numerik untuk analisis outlier
numeric_cols_data1 = data1.select_dtypes(include=[np.number]).columns.tolist()
numeric_cols_data2 = data2.select_dtypes(include=[np.number]).columns.tolist()

# --- Penerapan Strategi Penanganan Outlier untuk Data1 ---
print("\n=== PENERAPAN STRATEGI PENANGANAN OUTLIER UNTUK DATA1 ===")

# Menyimpan jumlah baris asli
original_shape_data1 = data1.shape[0]

# 1. HAPUS OUTLIER (Deletion) - Diterapkan secara berurutan pada setiap kolom
print("\n--- 1. HAPUS OUTLIER (Deletion) ---")
for col in numeric_cols_data1:
    # Memastikan kolom memiliki variasi sebelum menghapus outlier
    if data1[col].dropna().std() > 0:
        data1_handled_delete = remove_outliers(data1_handled_delete, col)
        print(f"  Setelah hapus outlier di '{col}': Shape = {data1_handled_del
print(f"  Final shape setelah hapus semua outlier secara berurutan: {data1_har
# Menghitung persentase baris yang terhapus
rows_deleted_data1 = original_shape_data1 - data1_handled_delete.shape[0]
percent_deleted_data1 = (rows_deleted_data1 / original_shape_data1 * 100) if o
print(f"  Total baris terhapus: {rows_deleted_data1} ({percent_deleted_data1:.


# 2. CAPPING OUTLIER (Winsorizing) - Diterapkan pada salinan data asli
print("\n--- 2. CAPPING OUTLIER (Winsorizing) ---")
```

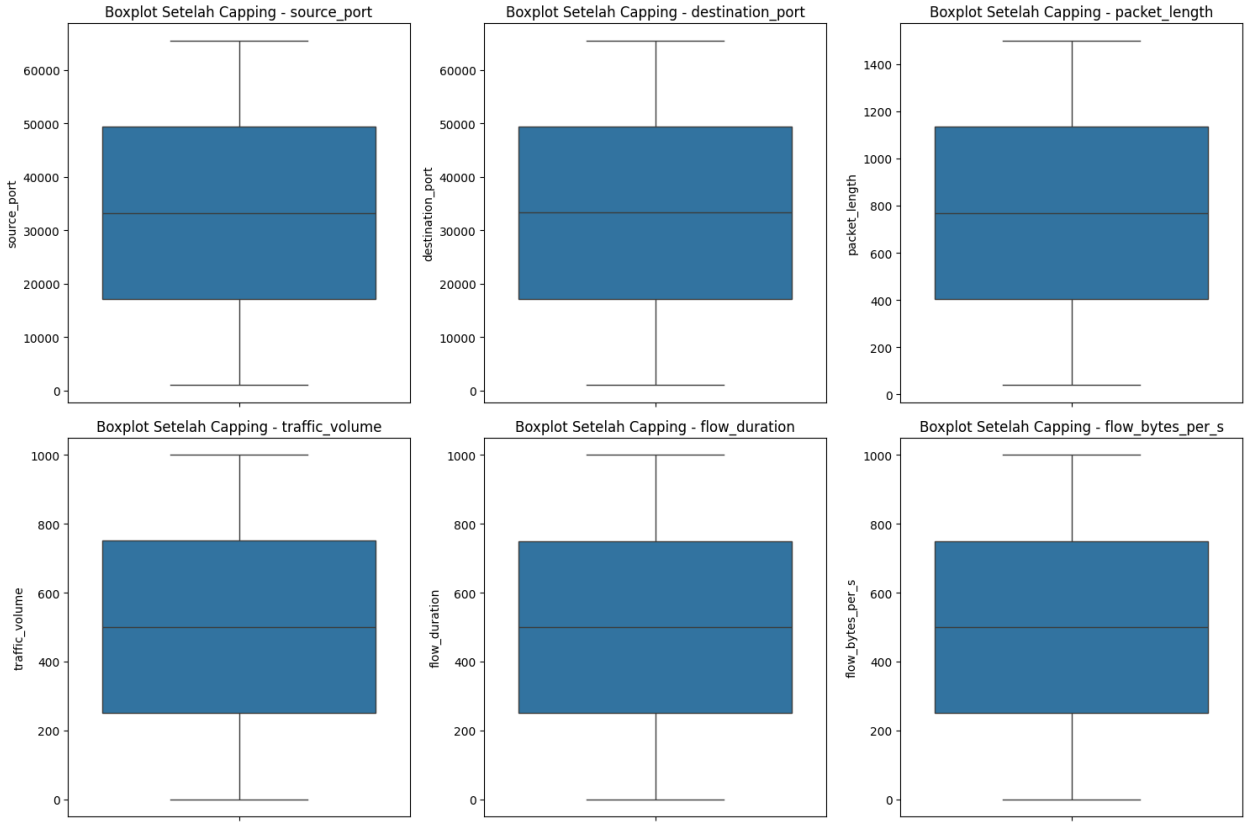
```python
# Iterasi melalui setiap kolom numerik untuk melakukan capping
for col in numeric_cols_data1:
     # Memastikan kolom memiliki variasi sebelum capping
    if data1[col].dropna().std() > 0:
        data1_handled_cap = cap_outliers(data1_handled_cap, col)
# Menampilkan shape data setelah capping (jumlah baris tetap sama)
print(f"  Shape setelah capping: {data1_handled_cap.shape} (tidak ada baris te


# 3. ISI DENGAN MEDIAN - Diterapkan pada salinan data asli
print("\n--- 3. ISI OUTLIER DENGAN MEDIAN ---")
# Iterasi melalui setiap kolom numerik untuk mengganti outlier dengan median
for col in numeric_cols_data1:
    # Memastikan kolom memiliki variasi sebelum mengisi outlier
    if data1[col].dropna().std() > 0:
        data1_handled_median = replace_outliers_with_median(data1_handled_medi
# Menampilkan shape data setelah mengisi outlier dengan median (jumlah baris t
print(f"  Shape setelah mengisi outlier dengan median: {data1_handled_median.s

# Opsional: Verifikasi Visualisasi (Boxplot) Setelah Penanganan Capping untuk
print("\n--- Verifikasi Visual (Boxplot Setelah Capping - Data1) ---")
plt.figure(figsize=(15, 10))
# Menampilkan boxplot untuk beberapa kolom pertama setelah capping
for i, col in enumerate(numeric_cols_data1[:6], 1): # Hanya menampilkan 6 kolo
    plt.subplot(2, 3, i) # Membuat subplot dalam grid 2x3
    sns.boxplot(y=data1_handled_cap[col].dropna()) # Membuat boxplot
    plt.title(f'Boxplot Setelah Capping - {col}') # Judul subplot
plt.tight_layout() # Menyesuaikan layout agar tidak tumpang tindih
plt.show() # Menampilkan plot


# --- Penerapan Strategi Penanganan Outlier untuk Data2 ---
print("\n=== PENERAPAN STRATEGI PENANGANAN OUTLIER UNTUK DATA2 ===")

# Menyimpan jumlah baris asli
original_shape_data2 = data2.shape[0]

# 1. HAPUS OUTLIER (Deletion) - Diterapkan secara berurutan pada setiap kolom
print("\n--- 1. HAPUS OUTLIER (Deletion) ---")
for col in numeric_cols_data2:
    # Memastikan kolom memiliki variasi sebelum menghapus outlier
    if data2[col].dropna().std() > 0:
        data2_handled_delete = remove_outliers(data2_handled_delete, col)
        print(f"  Setelah hapus outlier di '{col}': Shape = {data2_handled_del
print(f"  Final shape setelah hapus semua outlier secara berurutan: {data2_har
# Menghitung persentase baris yang terhapus
rows_deleted_data2 = original_shape_data2 - data2_handled_delete.shape[0]
percent_deleted_data2 = (rows_deleted_data2 / original_shape_data2 * 100) if o
print(f"  Total baris terhapus: {rows_deleted_data2} ({percent_deleted_data2:.

# 2. CAPPING OUTLIER (Winsorizing) - Diterapkan pada salinan data asli
print("\n--- 2. CAPPING OUTLIER (Winsorizing) ---")
# Iterasi melalui setiap kolom numerik untuk melakukan capping
```

```
for col in numeric_cols_data2:
    # Memastikan kolom memiliki variasi sebelum capping
    if data2[col].dropna().std() > 0:
        data2_handled_cap = cap_outliers(data2_handled_cap, col)
# Menampilkan shape data setelah capping (jumlah baris tetap sama)
print(f"  Shape setelah capping: {data2_handled_cap.shape} (tidak ada baris te

# 3. ISI DENGAN MEDIAN - Diterapkan pada salinan data asli
print("\n--- 3. ISI OUTLIER DENGAN MEDIAN ---")
# Iterasi melalui setiap kolom numerik untuk mengganti outlier dengan median
for col in numeric_cols_data2:
    # Memastikan kolom memiliki variasi sebelum mengisi outlier
    if data2[col].dropna().std() > 0:
        data2_handled_median = replace_outliers_with_median(data2_handled_medi
# Menampilkan shape data setelah mengisi outlier dengan median (jumlah baris t
print(f"  Shape setelah mengisi outlier dengan median: {data2_handled_median.s


# Opsional: Verifikasi Visualisasi (Boxplot) Setelah Penanganan Capping untuk
print("\n--- Verifikasi Visual (Boxplot Setelah Capping - Data2) ---")
plt.figure(figsize=(15, 10))
# Menampilkan boxplot untuk beberapa kolom pertama setelah capping
for i, col in enumerate(numeric_cols_data2[:6], 1): # Hanya menampilkan 6 kolo
    plt.subplot(2, 3, i) # Membuat subplot dalam grid 2x3
    sns.boxplot(y=data2_handled_cap[col].dropna()) # Membuat boxplot
    plt.title(f'Boxplot Setelah Capping - {col}') # Judul subplot
plt.tight_layout() # Menyesuaikan layout agar tidak tumpang tindih
plt.show() # Menampilkan plot


# Opsional: Transformasi Log untuk kolom skewed (jika data >0, contoh untuk ko
print("\n--- 4. TRANSFORMASI LOG (Opsional untuk Skewed Data) ---")

# Contoh: Terapkan log pada kolom 'traffic_volume' di data1 jika >0
if 'traffic_volume' in numeric_cols_data1:
    data1_log = data1.copy()
    # Pastikan kolom adalah numerik sebelum transformasi log
    if pd.api.types.is_numeric_dtype(data1_log['traffic_volume']):
        data1_log['traffic_volume_log'] = np.log1p(data1_log['traffic_volume']
        plt.figure(figsize=(12, 4))
        plt.subplot(1, 2, 1)
        sns.boxplot(y=data1['traffic_volume'].dropna())
        plt.title('Sebelum Log Transform')
        plt.subplot(1, 2, 2)
        sns.boxplot(y=data1_log['traffic_volume_log'].dropna())
        plt.title('Setelah Log Transform')
        plt.tight_layout()
        plt.show()
    else:
        print("Kolom 'traffic_volume' bukan tipe numerik, tidak bisa terapkan


# Ringkasan Statistik Setelah Penanganan (contoh untuk capping)
```

```
print("\n=== RINGKASAN STATISTIK SETELAH PENANGANAN (Capping) ===")
print("Data1 setelah capping:")
display(data1_handled_cap[numeric_cols_data1].describe().round(2))
print("\nData2 setelah capping:")
display(data2_handled_cap[numeric_cols_data2].describe().round(2))


# Visualisasi efek penanganan outlier pada Data2 menggunakan Violin Plots
print("\n=== VISUALISASI EFEK PENANGANAN OUTLIER PADA DATA2 (VIOLIN PLOT) ==="

for col in numeric_cols_data2:
    if data2[col].dropna().std() > 0:  # Hanya untuk kolom dengan variasi
        plt.figure(figsize=(15, 5))

        # Create a temporary DataFrame for plotting
        plot_df = pd.DataFrame({
            'Nilai': pd.concat([
                data2[col].rename('Sebelum Penanganan'),
                remove_outliers(data2.copy(), col)[col].rename('Setelah Hapus
                cap_outliers(data2.copy(), col)[col].rename('Setelah Capping')
            ]),
            'Kondisi': pd.concat([
                pd.Series('Sebelum Penanganan', index=data2[col].index),
                pd.Series('Setelah Hapus Outlier', index=remove_outliers(data2
                pd.Series('Setelah Capping', index=cap_outliers(data2.copy(),
            ])
        })

        sns.violinplot(x='Kondisi', y='Nilai', data=plot_df.dropna())
        plt.title(f'Efek Penanganan Outlier pada "{col}" - Violin Plot')
        plt.xlabel("Kondisi Penanganan")
        plt.ylabel(col)
        plt.tight_layout()
        plt.show()
```

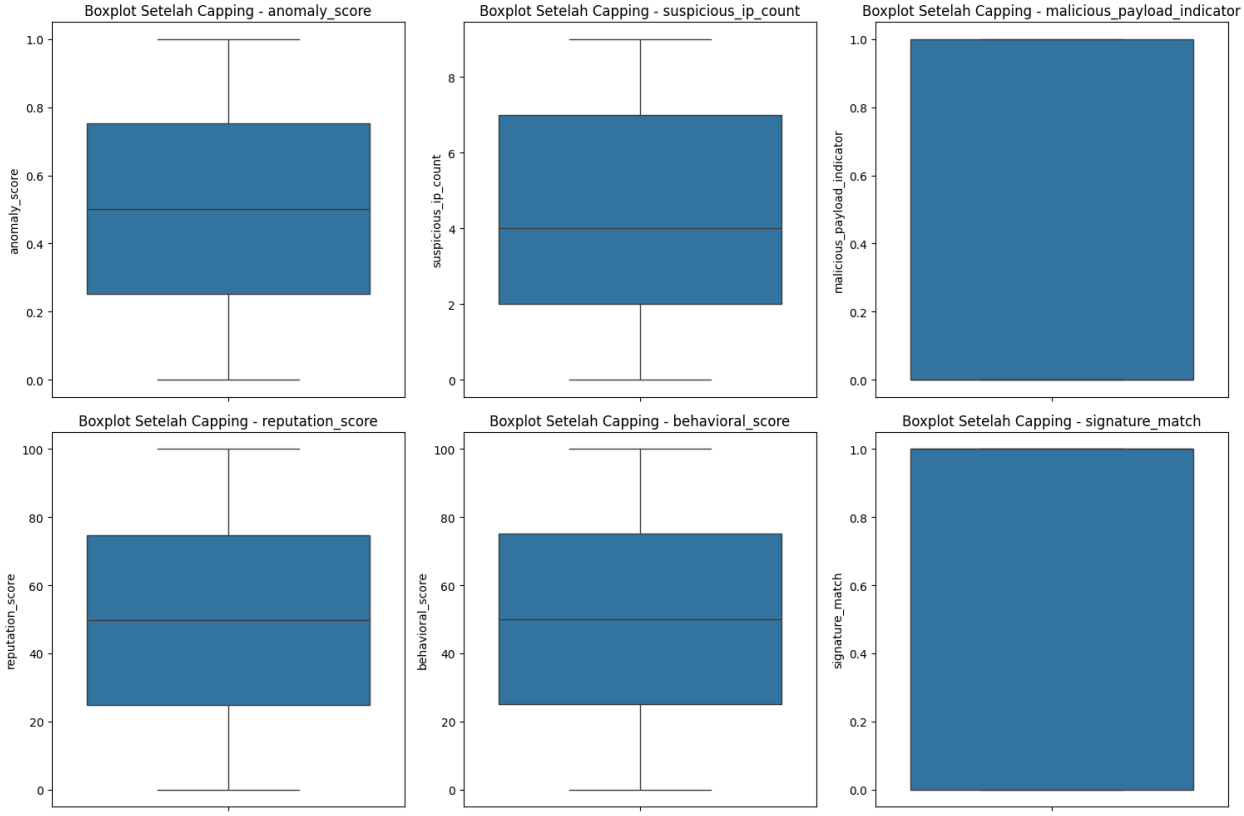
```
=== PENERAPAN STRATEGI PENANGANAN OUTLIER UNTUK DATA1 ===

--- 1. HAPUS OUTLIER (Deletion) ---
  Setelah hapus outlier di 'source_port': Shape = (200000, 34)
  Setelah hapus outlier di 'destination_port': Shape = (200000, 34)
  Setelah hapus outlier di 'packet_length': Shape = (200000, 34)
  Setelah hapus outlier di 'traffic_volume': Shape = (200000, 34)
  Setelah hapus outlier di 'flow_duration': Shape = (200000, 34)
  Setelah hapus outlier di 'flow_bytes_per_s': Shape = (200000, 34)
  Setelah hapus outlier di 'flow_packets_per_s': Shape = (200000, 34)
  Setelah hapus outlier di 'packet_count': Shape = (200000, 34)
  Setelah hapus outlier di 'average_packet_size': Shape = (200000, 34)
  Setelah hapus outlier di 'min_packet_size': Shape = (200000, 34)
  Setelah hapus outlier di 'max_packet_size': Shape = (200000, 34)
  Setelah hapus outlier di 'packet_size_variance': Shape = (200000, 34)
  Setelah hapus outlier di 'header_length': Shape = (200000, 34)
  Setelah hapus outlier di 'payload_length': Shape = (200000, 34)
  Setelah hapus outlier di 'ip_ttl': Shape = (200000, 34)
  Setelah hapus outlier di 'tcp_window_size': Shape = (200000, 34)
  Setelah hapus outlier di 'dns_query_count': Shape = (200000, 34)
  Setelah hapus outlier di 'dns_response_count': Shape = (200000, 34)
  Setelah hapus outlier di 'http_status_code': Shape = (200000, 34)
  Setelah hapus outlier di 'year': Shape = (200000, 34)
  Setelah hapus outlier di 'month': Shape = (200000, 34)
  Setelah hapus outlier di 'day': Shape = (200000, 34)
  Setelah hapus outlier di 'hour': Shape = (200000, 34)
  Final shape setelah hapus semua outlier secara berurutan: (200000, 34)
  Total baris terhapus: 0 (0.00%)

--- 2. CAPPING OUTLIER (Winsorizing) ---
```

```
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-31379.875' has dtype incompatible with int64, please explicitly cast to
a compatible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-31337.5' has dtype incompatible with int64, please explicitly cast to a
compatible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-126.5' has dtype incompatible with int64, please explicitly cast to a c
ompatible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-32837.125' has dtype incompatible with int64, please explicitly cast to
a compatible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-5.5' has dtype incompatible with int64, please explicitly cast to a com
patible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-5.5' has dtype incompatible with int64, please explicitly cast to a com
patible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-14.5' has dtype incompatible with int32, please explicitly cast to a co
mpatible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-14.5' has dtype incompatible with int32, please explicitly cast to a co
mpatible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
  Shape setelah capping: (200000, 34) (tidak ada baris terhapus)

--- 3. ISI OUTLIER DENGAN MEDIAN ---
  Shape setelah mengisi outlier dengan median: (200000, 34) (tidak ada baris te
rhapus)

--- Verifikasi Visual (Boxplot Setelah Capping - Data1) ---
```

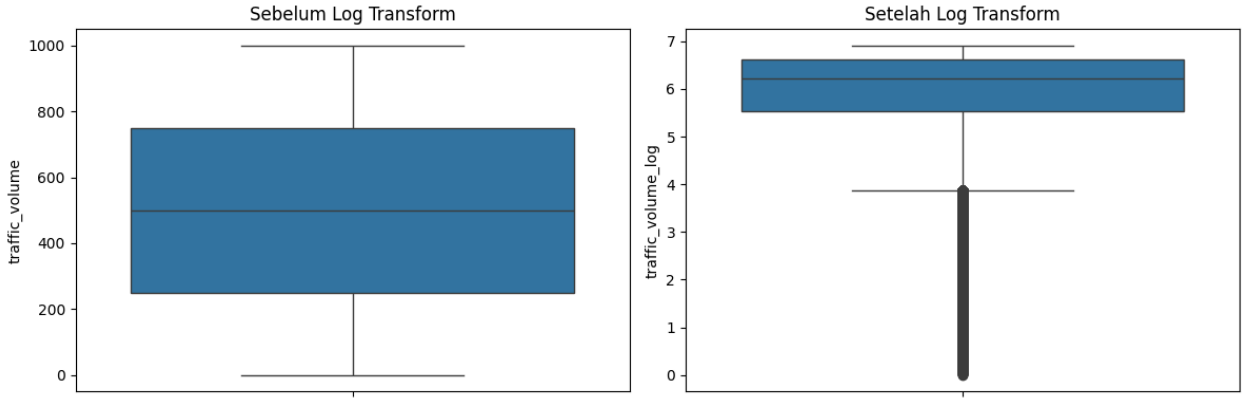



```
=== PENERAPAN STRATEGI PENANGANAN OUTLIER UNTUK DATA2 ===

--- 1. HAPUS OUTLIER (Deletion) ---
  Setelah hapus outlier di 'anomaly_score': Shape = (200000, 10)
  Setelah hapus outlier di 'suspicious_ip_count': Shape = (200000, 10)
  Setelah hapus outlier di 'malicious_payload_indicator': Shape = (200000, 10)
  Setelah hapus outlier di 'reputation_score': Shape = (200000, 10)
  Setelah hapus outlier di 'behavioral_score': Shape = (200000, 10)
  Setelah hapus outlier di 'signature_match': Shape = (200000, 10)
  Setelah hapus outlier di 'heuristic_score': Shape = (200000, 10)
  Final shape setelah hapus semua outlier secara berurutan: (200000, 10)
  Total baris terhapus: 0 (0.00%)

--- 2. CAPPING OUTLIER (Winsorizing) ---
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-5.5' has dtype incompatible with int64, please explicitly cast to a com
patible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-1.5' has dtype incompatible with int64, please explicitly cast to a com
patible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-1.5' has dtype incompatible with int64, please explicitly cast to a com
patible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
```

```
  Shape setelah capping: (200000, 10) (tidak ada baris terhapus)

--- 3. ISI OUTLIER DENGAN MEDIAN ---
  Shape setelah mengisi outlier dengan median: (200000, 10) (tidak ada baris te
rhapus)

--- Verifikasi Visual (Boxplot Setelah Capping - Data2) ---
```




```
--- 4. TRANSFORMASI LOG (Opsional untuk Skewed Data) ---
```




```
=== RINGKASAN STATISTIK SETELAH PENANGANAN (Capping) ===
Data1 setelah capping:
```

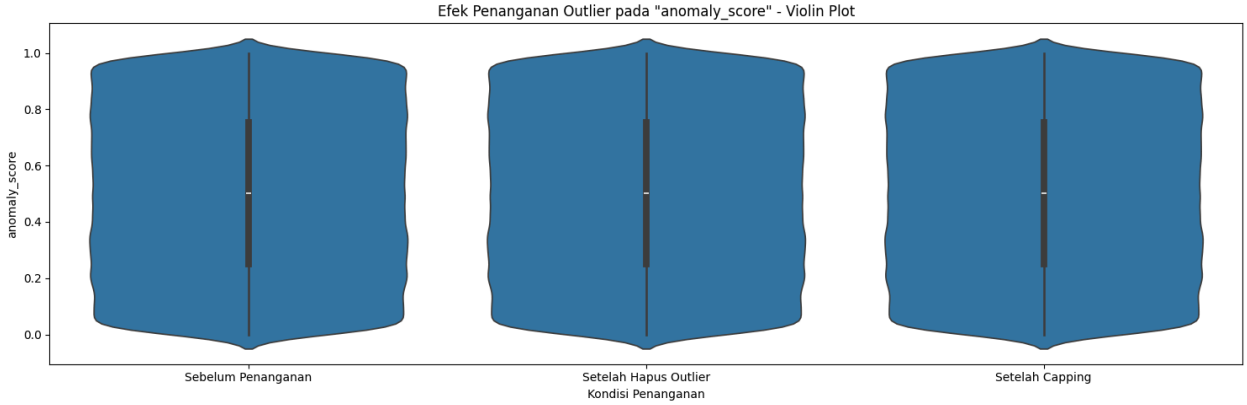
	source_port	destination_port	packet_length	traffic_volume	flow_duration
count	200000.00	200000.00	200000.00	200000.00	200000.00
mean	33227.25	33293.50	769.62	500.11	500.04
std	18627.85	18626.16	421.44	289.09	288.44
min	1024.00	1024.00	40.00	0.00	0.00
25%	17075.00	17126.00	405.00	249.78	250.31
50%	33174.00	33344.00	769.00	500.17	499.95
75%	49378.25	49435.00	1135.00	751.00	748.88
max	65534.00	65534.00	1499.00	1000.00	1000.00

8 rows × 23 columns

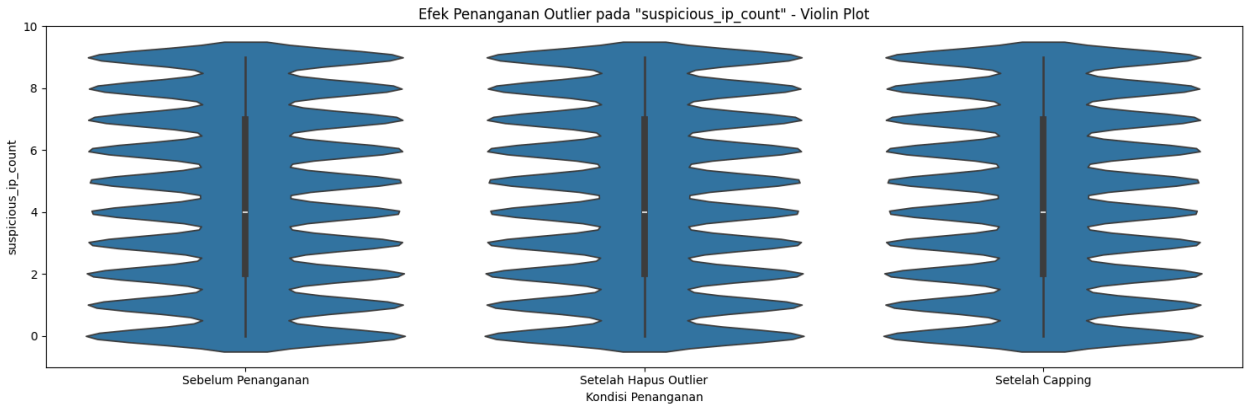
Data2 setelah capping:

	anomaly_score	suspicious_ip_count	malicious_payload_indicator	reputatio
count	200000.00	200000.00	200000.0	20
mean	0.50	4.49	0.5	
std	0.29	2.87	0.5	
min	0.00	0.00	0.0	
25%	0.25	2.00	0.0	
50%	0.50	4.00	0.0	
75%	0.75	7.00	1.0	
max	1.00	9.00	1.0	

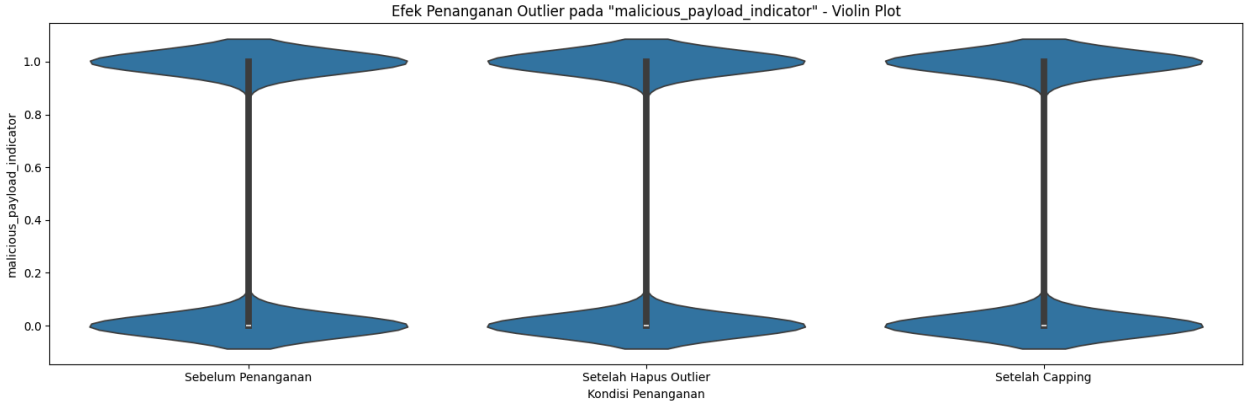
=== VISUALISASI EFEK PENANGANAN OUTLIER PADA DATA2 (VIOLIN PLOT) ===

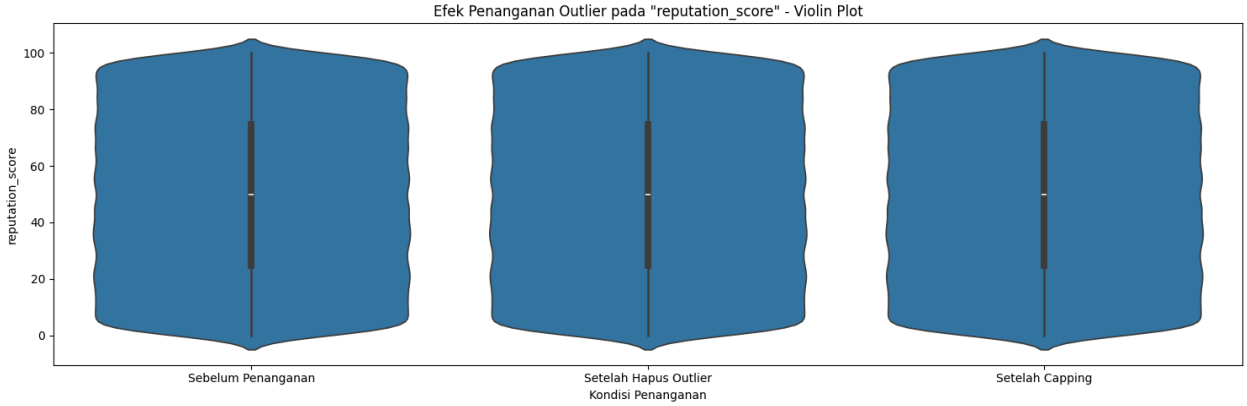


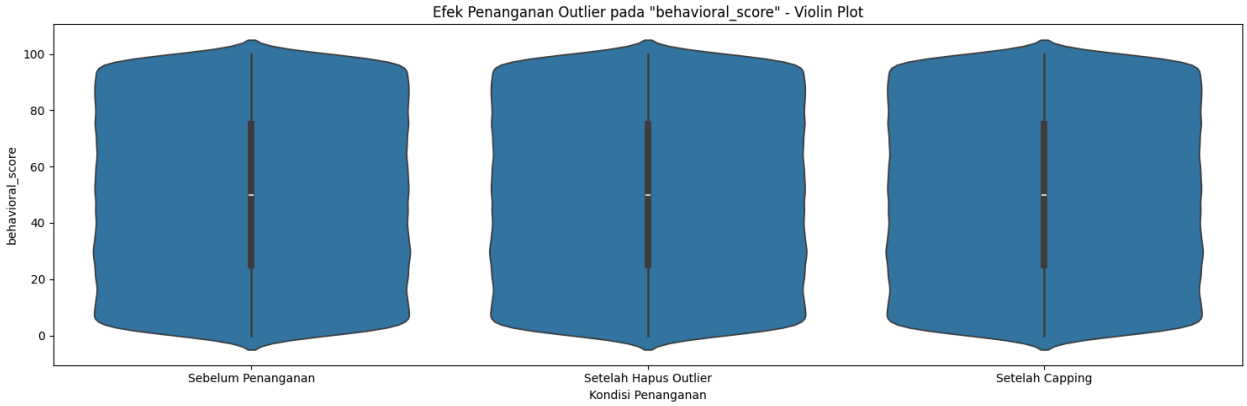
```
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-5.5' has dtype incompatible with int64, please explicitly cast to a com
patible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-5.5' has dtype incompatible with int64, please explicitly cast to a com
patible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
```



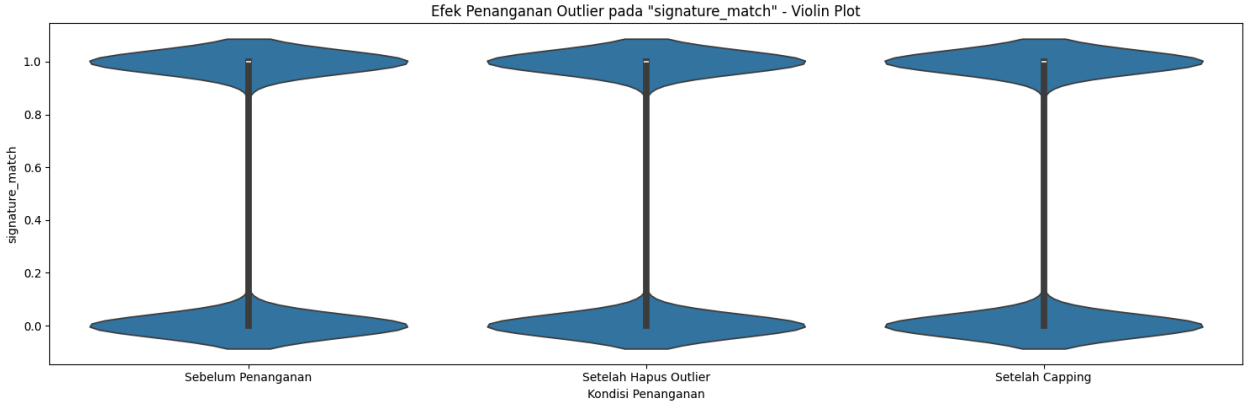

```
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-1.5' has dtype incompatible with int64, please explicitly cast to a com
patible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-1.5' has dtype incompatible with int64, please explicitly cast to a com
patible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
```

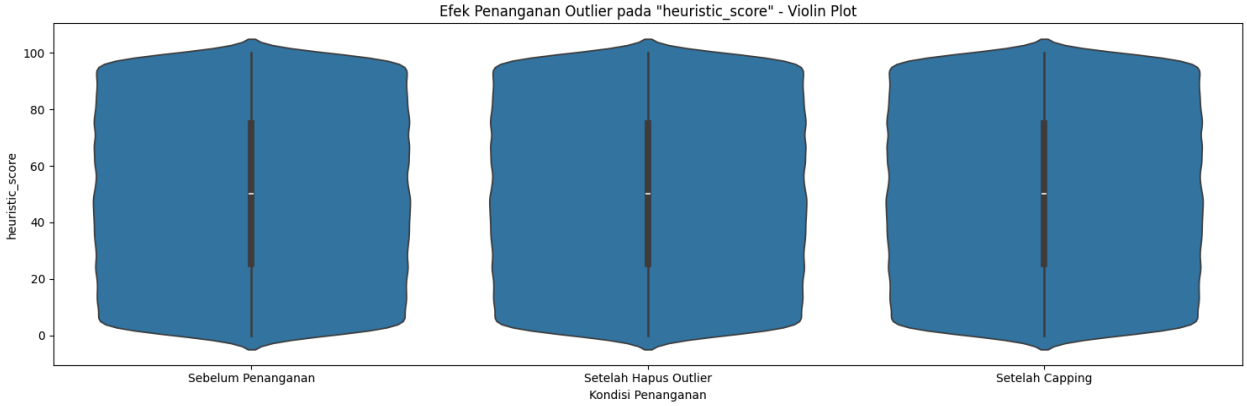







```
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-1.5' has dtype incompatible with int64, please explicitly cast to a com
patible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
/tmp/ipython-input-4132717762.py:33: FutureWarning: Setting an item of incompat
ible dtype is deprecated and will raise an error in a future version of pandas.
Value '-1.5' has dtype incompatible with int64, please explicitly cast to a com
patible dtype first.
  df_cap.loc[df_cap[col] < lower, col] = lower
```





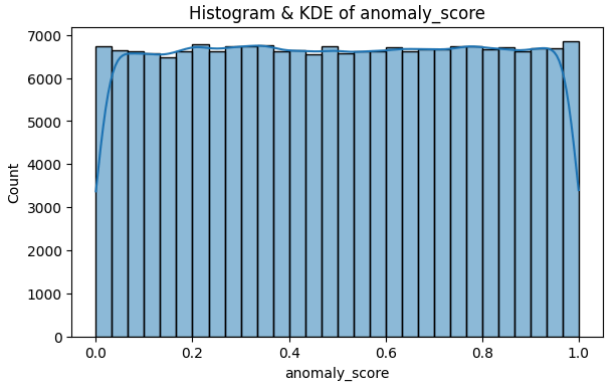
Strategi penanganan outlier (Hapus, Capping, Imputasi)

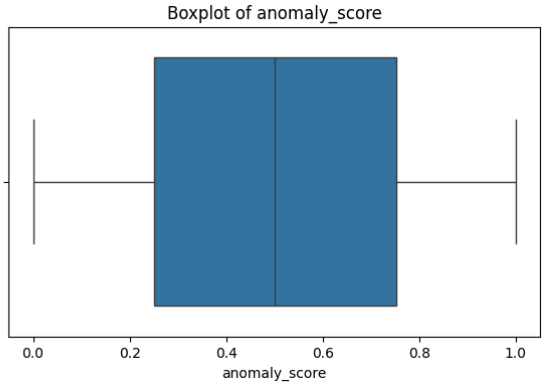
3. Univariate Analysis

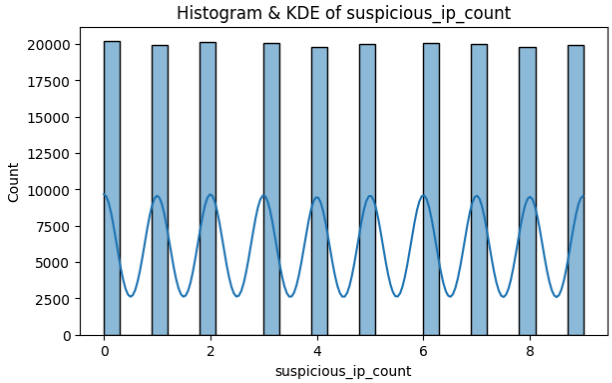
- Distribusi variabel numerik (histogram, KDE plot, boxplot)
- Distribusi variabel kategorikal (barplot, pie chart, frekuensi)
- Statistik ringkas per variabel

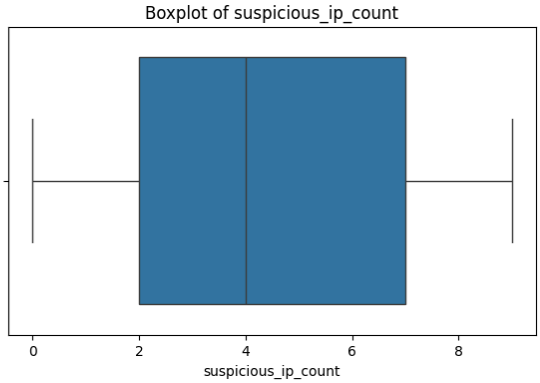
In [40]:

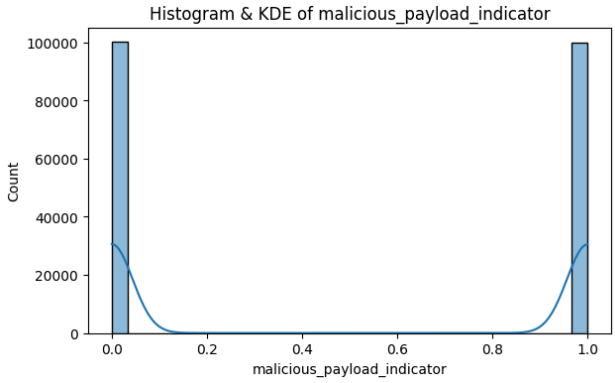
```
import matplotlib.pyplot as plt
import seaborn as sns

# --- untuk malware_df ---
numeric_cols = data2.select_dtypes(include=['int64', 'float64']).columns

for col in numeric_cols:
    plt.figure(figsize=(15,4))

    # Histogram + KDE
    plt.subplot(1,2,1)
    sns.histplot(data2[col], kde=True, bins=30)
    plt.title(f"Histogram & KDE of {col}")

    # Boxplot
    plt.subplot(1,2,2)
    sns.boxplot(x=data2[col])
    plt.title(f"Boxplot of {col}")

    plt.show()
```

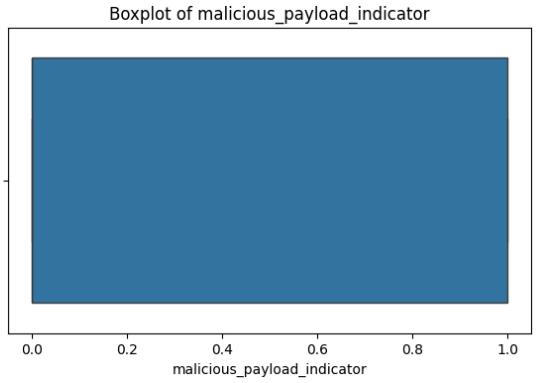

Histogram & KDE of anomaly_score
Count
anomaly_score

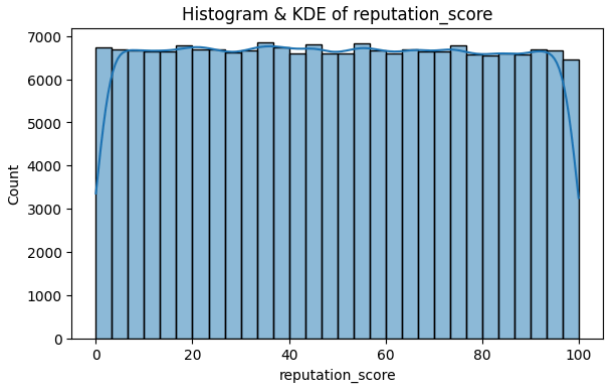

Boxplot of anomaly_score
anomaly_score

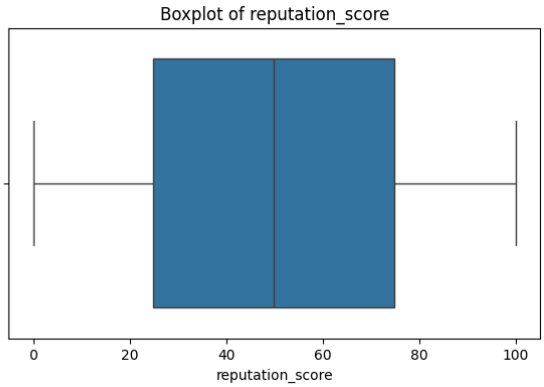

Histogram & KDE of suspicious_ip_count
Count
suspicious_ip_count

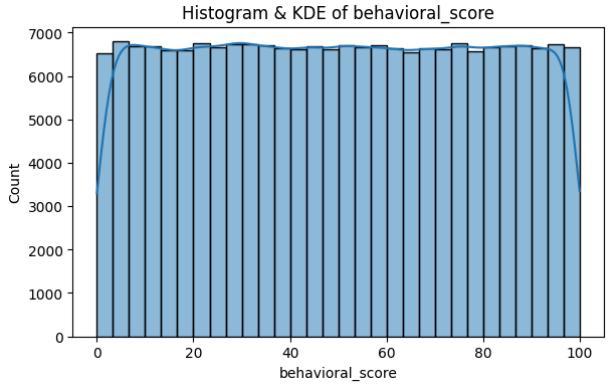

Boxplot of suspicious_ip_count
suspicious_ip_count

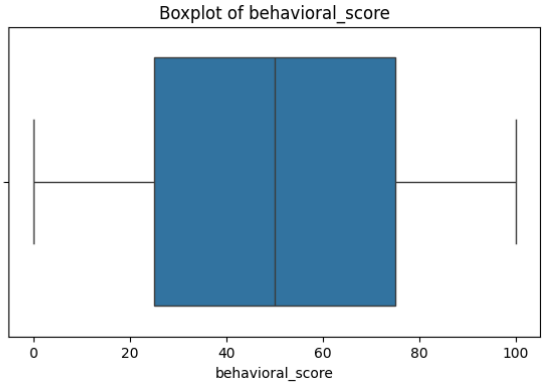

Histogram & KDE of malicious_payload_indicator
Count
malicious_payload_indicator

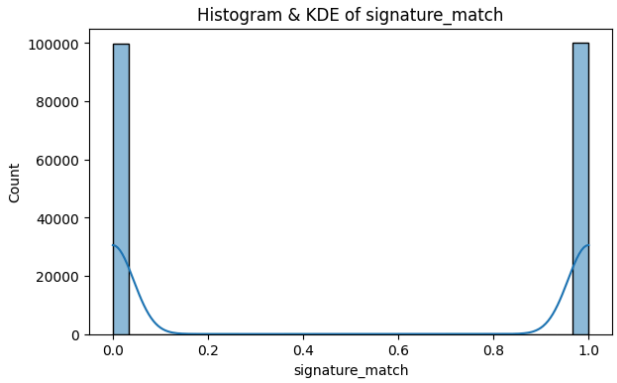

Boxplot of malicious_payload_indicator
malicious_payload_indicator

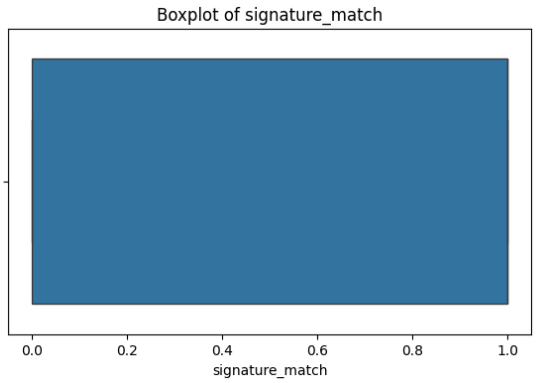

Histogram & KDE of reputation_score
Count
reputation_score

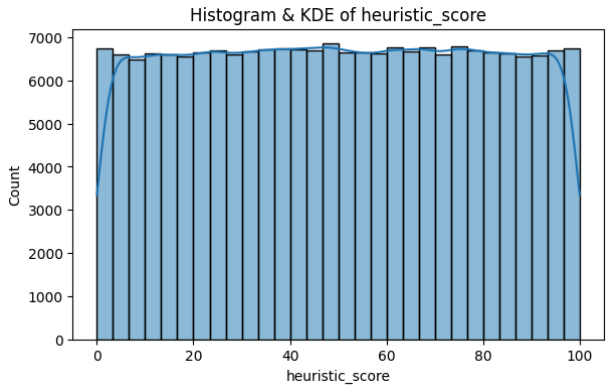

Boxplot of reputation_score
reputation_score

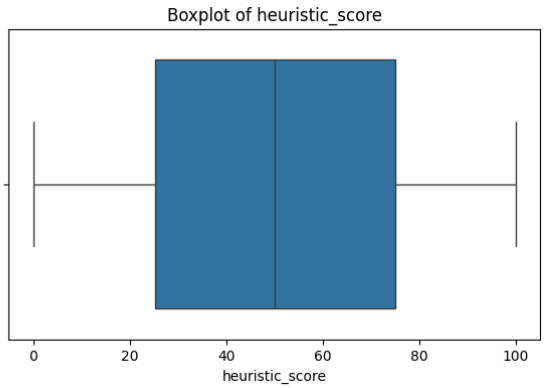








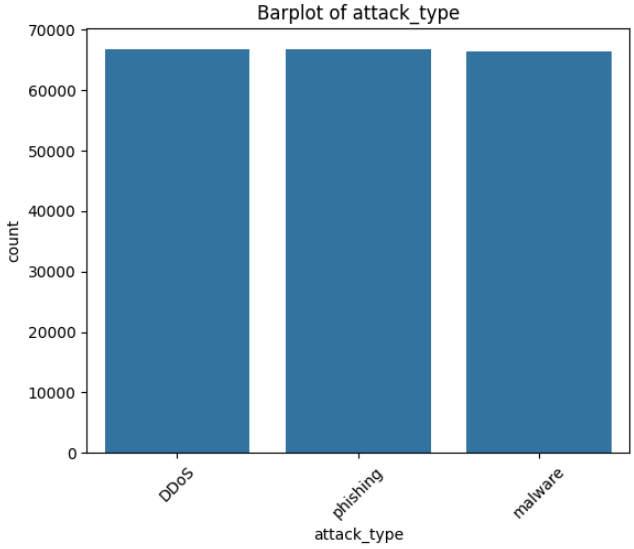


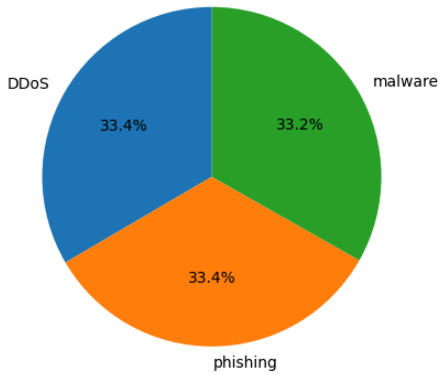


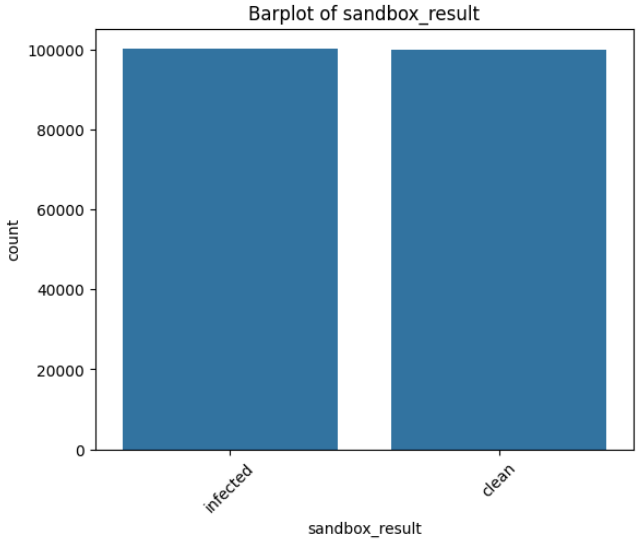
In [41]:

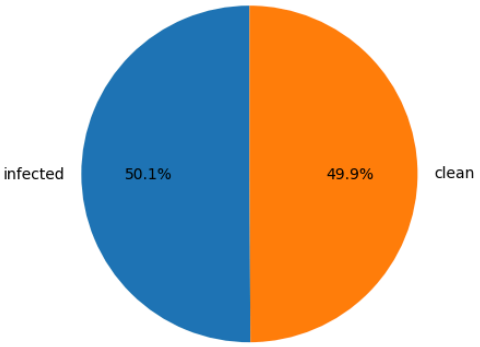
```
# --- untuk malware_df ---
categorical_cols = data2.select_dtypes(include=['object']).columns

for col in categorical_cols:
    plt.figure(figsize=(14,5))

    # Barplot
    plt.subplot(1,2,1)
    sns.countplot(x=data2[col], order=data2[col].value_counts().index)
    plt.title(f"Barplot of {col}")
    plt.xticks(rotation=45)

    # Pie chart
    plt.subplot(1,2,2)
    data2[col].value_counts().plot.pie(autopct='%1.1f%%', startangle=90)
    plt.title(f"Pie chart of {col}")
    plt.ylabel("")
```

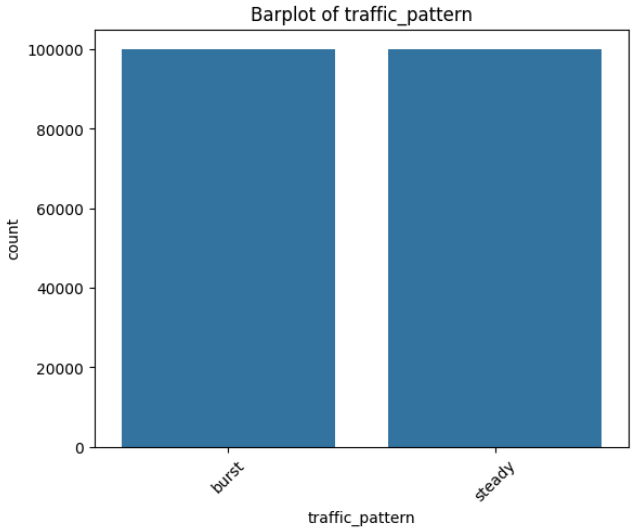
```
    plt.show()
```

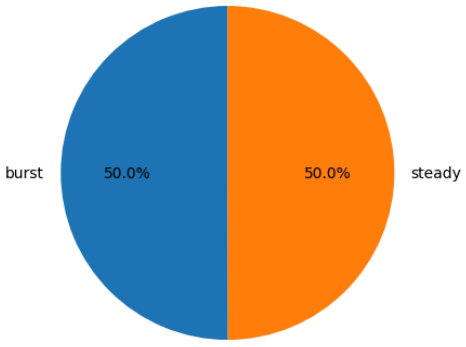












In [42]:
```
# Statistik numerik
```

```
print("📊 Statistik Ringkas (Numerik)")
display(data2.describe().T)

# Statistik kategorikal
print("\n📊 Statistik Ringkas (Kategorikal)")
for col in categorical_cols:
    print(f"\nKolom: {col}")
    print(data2[col].value_counts())
```

📊 Statistik Ringkas (Numerik)

	count	mean	std	min	25%
anomaly_score	200000.0	0.500966	0.288896	0.000002	0.251379
suspicious_ip_count	200000.0	4.489850	2.873703	0.000000	2.000000
malicious_payload_indicator	200000.0	0.499120	0.500000	0.000000	0.000000
reputation_score	200000.0	49.849355	28.827126	0.000284	24.860720
behavioral_score	200000.0	49.998517	28.876260	0.000287	25.058753
signature_match	200000.0	0.500295	0.500001	0.000000	0.000000
heuristic_score	200000.0	50.074518	28.816467	0.000678	25.195372

📊 Statistik Ringkas (Kategorikal)

```
Kolom: attack_type
attack_type
DDoS        66864
phishing    66751
malware     66385
Name: count, dtype: int64

Kolom: sandbox_result
sandbox_result
infected    100185
clean        99815
Name: count, dtype: int64

Kolom: traffic_pattern
traffic_pattern
burst     100065
steady     99935
Name: count, dtype: int64
```

Untuk memahami karakteristik data, analisis dilakukan melalui tiga tahap utama. Pertama, variabel numerik divisualisasikan dengan histogram, KDE plot, dan boxplot untuk melihat bentuk distribusi serta mendeteksi adanya outlier. Kedua, variabel kategorikal dianalisis dengan barplot dan pie chart untuk mengetahui frekuensi dan proporsi tiap kategori, seperti jenis serangan atau protokol. Ketiga, dihitung statistik ringkas berupa nilai minimum, maksimum, rata-rata, kuartil, dan distribusi frekuensi kategori. Hasil analisis ini memberikan gambaran umum pola

data dan menjadi dasar identifikasi pola serangan jaringan.

4. Bivariate Analysis

- Numerik vs numerik: scatter plot, korelasi
- Numerik vs kategorikal: boxplot, violin plot, uji t/ANOVA
- Kategorikal vs kategorikal: crosstab, chi-square test, stacked barplot

Merging Data1 and Data2

In [43]:
```
# Merge data1 and data2 based on index
# Assuming the rows in both dataframes correspond to the same observations
data_merged = pd.merge(data1, data2, left_index=True, right_index=True)

# Display the first few rows of the merged dataframe
display(data_merged.head())

# Check the shape of the merged dataframe
print("\nShape of merged dataframe:", data_merged.shape)
```

	timestamp	source_ip	destination_ip	source_port	destination_port	prot
0	2024-06-28 19:02:55	86.230.134.129	3.46.98.34	3756	35357	I
1	2022-12-09 13:36:47	37.211.177.132	40.109.190.27	44591	30823	I
2	2022-08-22 04:59:38	54.129.79.47	77.250.43.217	43637	60156	
3	2023-02-07 15:30:41	157.59.116.46	211.120.32.204	53440	17944	
4	2020-12-08 21:06:49	161.125.154.101	62.15.8.89	46915	3390	I

5 rows × 44 columns

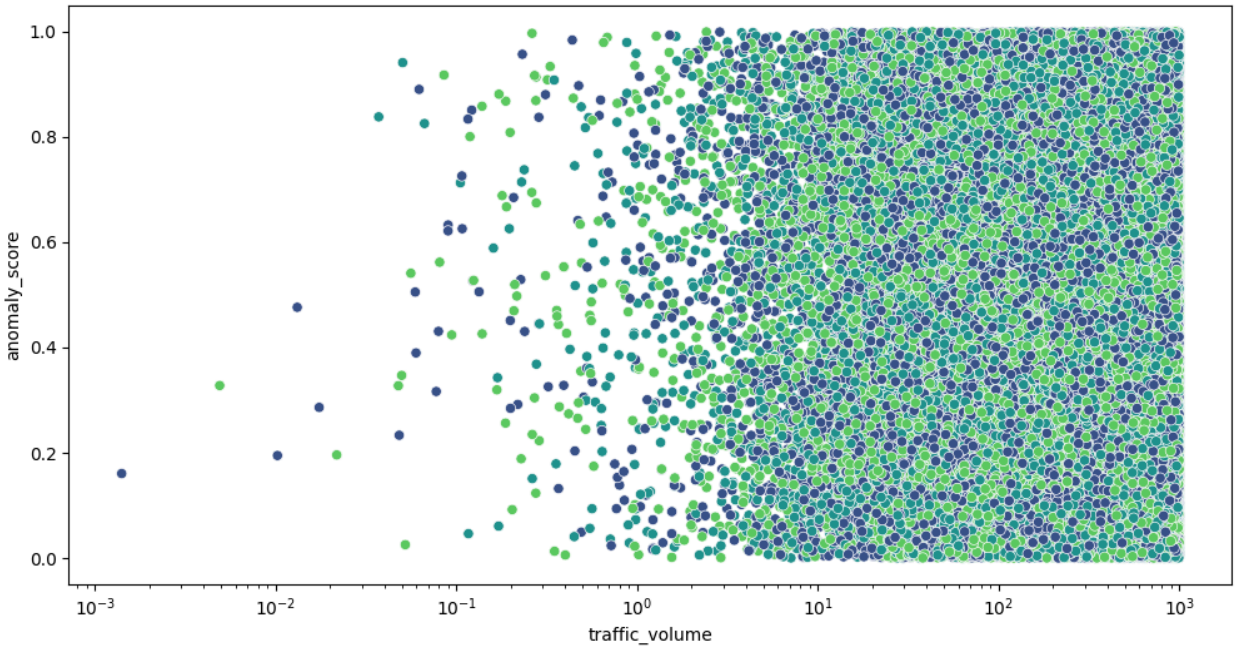
```
Shape of merged dataframe: (200000, 44)
```

4.1 Numerik vs numerik: scatter plot, korelasi

Data1 (Network)	Data2 (Malware)	Alasan Hubungan
`traffic_volume`	`threat_score`	Lalu lintas yang lebih besar dapat meningkatkan potensi ancaman.

4.1.1 Korelasi Traffic Volume Jaringan dan Skor Ancaman Malware

In [44]:
```
# Scatter plot dengan skala log
plt.figure(figsize=(12, 6))
sns.scatterplot(
    x='traffic_volume',
    y='anomaly_score',
    hue='attack_type',
    data=data_merged,
    palette='viridis',
    legend=False
)
plt.xscale('log')
plt.show()
```



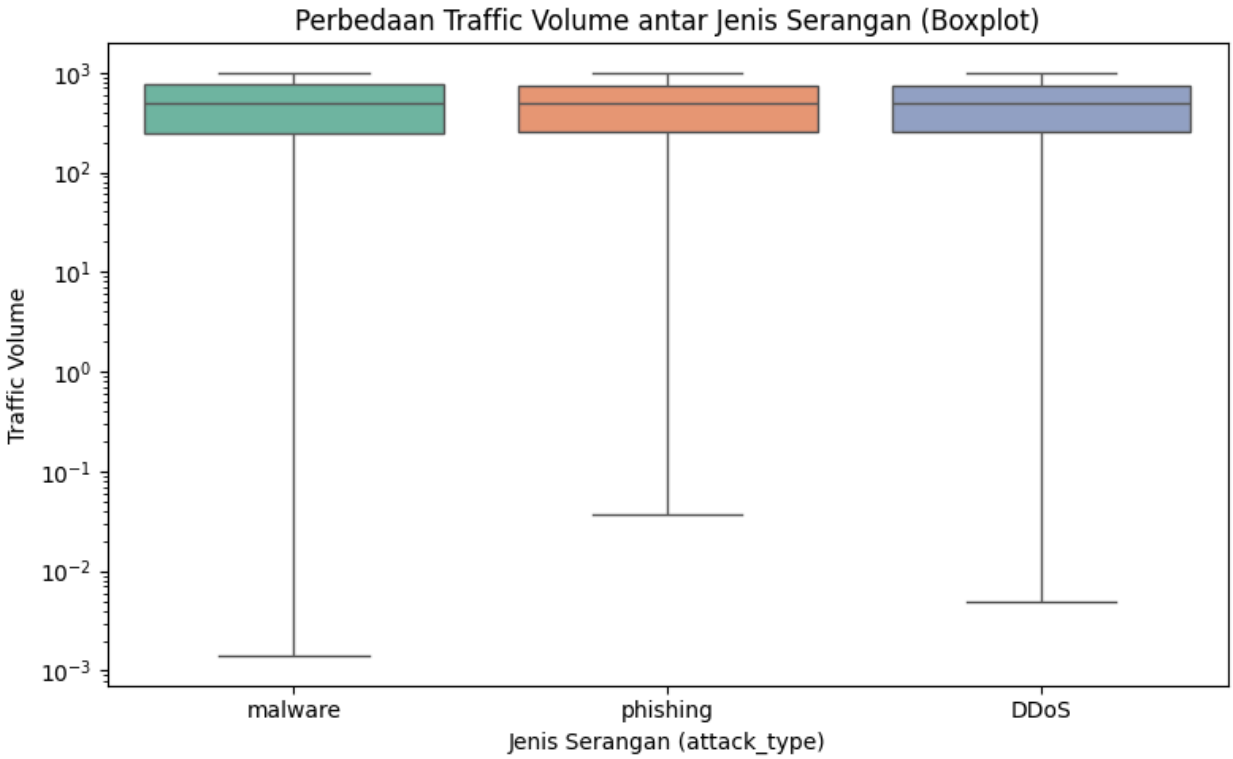
Plot menunjukkan bahwa volume lalu lintas jaringan (traffic_volume) dan skor anomali (anomaly_score) tidak memiliki hubungan linear yang jelas, karena setiap jenis serangan—malware, phishing, maupun DDoS—tersebar di seluruh rentang skor anomali. Namun, pola yang tampak adalah serangan DDoS cenderung lebih banyak muncul pada traffic volume tinggi, sesuai karakteristiknya yang menghasilkan lalu lintas besar, sedangkan malware dan phishing muncul di berbagai tingkat trafik. Hal ini menegaskan bahwa volume trafik saja tidak cukup untuk membedakan jenis serangan sehingga diperlukan fitur tambahan untuk analisis deteksi yang lebih akurat.

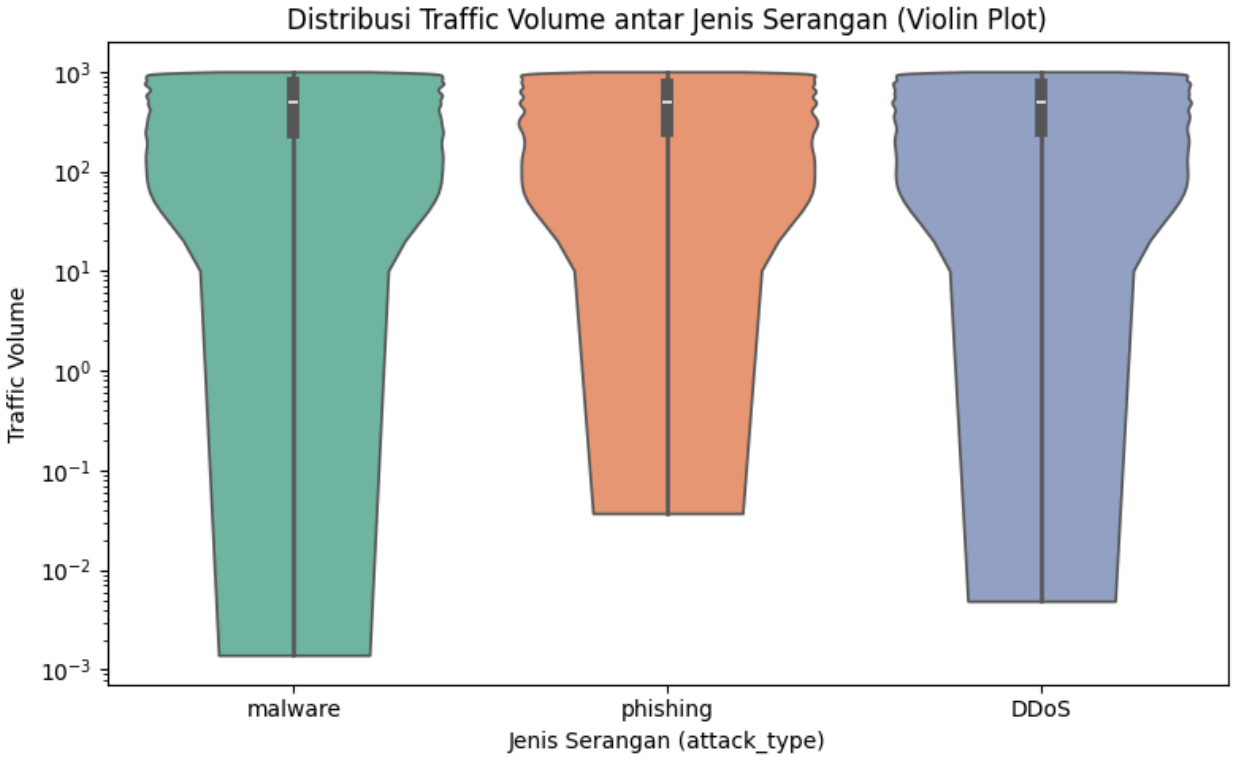
4.2 Numerik vs kategorikal: boxplot, violin plot, uji t/ANOVA

Data1 (Numerik)	Data2 (Kategorikal)	Alasan Hubungan
`traffic_volume`	`attack_type`	Volume traffic mungkin berbeda antar jenis serangan (DoS, malware, dll).

4.2.1 Distribusi Volume Lalu Lintas per Jenis Serangan

In [45]:
```
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats

# --- Boxplot ---
plt.figure(figsize=(8,5))
sns.boxplot(
    x='attack_type',
    y='traffic_volume',
    hue='attack_type',      # <--- tambahkan
    legend=False,           # <--- matikan legend supaya tidak dobel
    data=data_merged,
    palette='Set2'
)
plt.yscale('log')       # skala log agar distribusi traffic terlihat lebih jel
plt.title("Perbedaan Traffic Volume antar Jenis Serangan (Boxplot)")
plt.xlabel("Jenis Serangan (attack_type)")
plt.ylabel("Traffic Volume")
plt.tight_layout()
plt.show()

# --- Violin plot ---
plt.figure(figsize=(8,5))
sns.violinplot(
    x='attack_type',
    y='traffic_volume',
    hue='attack_type',   # <-- tambahkan
    data=data_merged,
    palette='Set2',
    legend=False,        # hilangkan legend ganda
    cut=0
)
plt.yscale('log')
plt.title("Distribusi Traffic Volume antar Jenis Serangan (Violin Plot)")
plt.xlabel("Jenis Serangan (attack_type)")
plt.ylabel("Traffic Volume")
plt.tight_layout()
plt.show()
```





Kedua grafik (boxplot dan violin plot) menunjukkan distribusi traffic volume pada tiga jenis serangan berbeda (malware, phishing, dan DDoS). Boxplot memperlihatkan ringkasan statistik (median, kuartil, dan outlier) di mana terlihat bahwa ketiga jenis serangan memiliki rentang distribusi yang lebar dengan sebagian besar data terkonsentrasi di traffic volume tinggi (sekitar ratusan hingga

ribuan). Sementara itu, violin plot memberikan gambaran lebih detail tentang bentuk distribusi data: terlihat bahwa distribusi traffic volume pada setiap jenis serangan cenderung mirip dengan konsentrasi besar di nilai tinggi, tetapi tetap ada variasi yang luas di nilai rendah hingga ekstrem. Dengan demikian, meskipun ada sedikit variasi antar jenis serangan, secara umum ketiganya menunjukkan pola distribusi traffic volume yang hampir sama, sehingga sulit menyimpulkan bahwa jenis serangan tertentu selalu menghasilkan traffic volume yang lebih besar atau lebih kecil dibandingkan lainnya.

4.3 Kategorikal vs kategorikal: crosstab, chi-square test, stacked barplot

Data1 (Numerik)	Data2 (Kategorikal)	Alasan Hubungan
`traffic_volume`	`attack_type`	Volume traffic mungkin berbeda antar jenis serangan (DoS, malware, dll).

4.3.1 Distribusi Jenis Protokol pada Berbagai Tipe Serangan

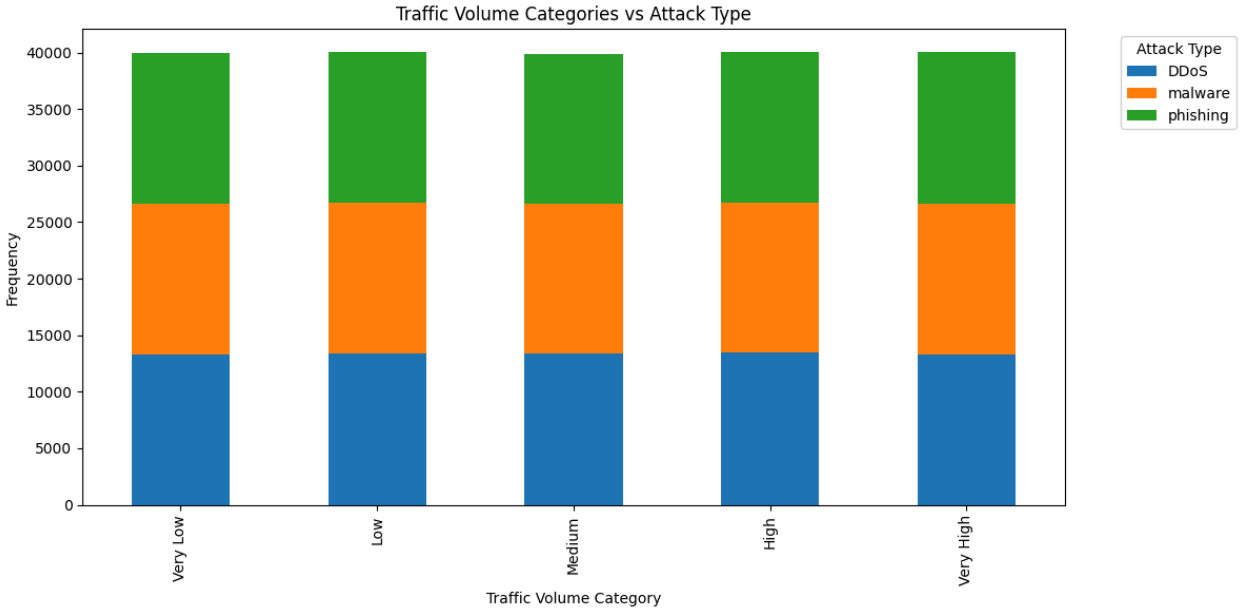
In [46]:
```
# Crosstab antara traffic_volume (dikategorikan) dan attack_type
# Pertama, kategorikan traffic_volume
data_merged['traffic_volume_category'] = pd.cut(data_merged['traffic_volume'],
                                                bins=5,
                                                labels=['Very Low', 'Low', 'Mec

# Buat crosstab
crosstab_result = pd.crosstab(data_merged['traffic_volume_category'],
                              data_merged['attack_type'])

print("Crosstab Traffic Volume vs Attack Type:")
display(crosstab_result)

# Stacked bar plot
plt.figure(figsize=(12, 6))
crosstab_result.plot(kind='bar', stacked=True, figsize=(12, 6))
plt.title('Traffic Volume Categories vs Attack Type')
plt.xlabel('Traffic Volume Category')
plt.ylabel('Frequency')
plt.legend(title='Attack Type', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()

# Chi-square test
from scipy.stats import chi2_contingency
chi2, p_value, dof, expected = chi2_contingency(crosstab_result)
print(f"\nChi-square Test Results:")
```

```
print(f"Chi2 statistic: {chi2:.4f}")
print(f"P-value: {p_value:.4f}")
print(f"Degrees of freedom: {dof}")
print(f"Significant association: {'Yes' if p_value < 0.05 else 'No'}")
```

```
Crosstab Traffic Volume vs Attack Type:
```

attack_type	DDoS	malware	phishing
traffic_volume_category			
Very Low	13280	13374	13345
Low	13387	13295	13375
Medium	13419	13174	13270
High	13468	13228	13370
Very High	13310	13314	13391

```
<Figure size 1200x600 with 0 Axes>
```



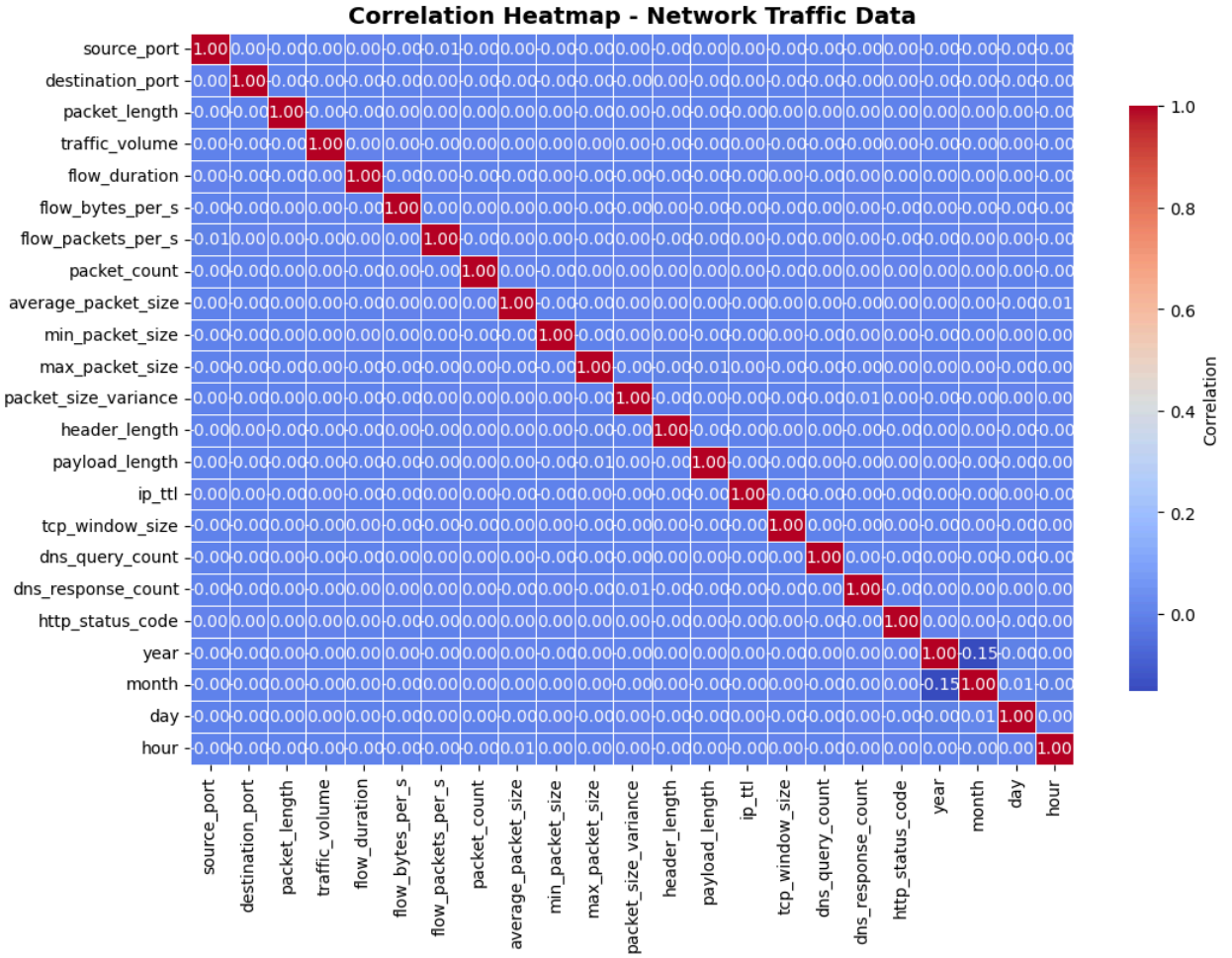

```
Chi-square Test Results:
Chi2 statistic: 3.6282
P-value: 0.8890
Degrees of freedom: 8
Significant association: No
```

5. Multivariate Analysis

- Heatmap korelasi
- Pairplot / scatter matrix
- PCA atau teknik dimensionality reduction (jika dataset besar)

In [47]:
```
# Heatmap korelasi dengan tampilan lebih bagus
```

```
plt.figure(figsize=(12,8))
sns.heatmap(
    data1.corr(numeric_only=True),
    cmap="coolwarm",
    annot=True,           # tampilkan nilai korelasi
    fmt=".2f",            # format angka 2 desimal
    linewidths=0.5,       # kasih garis pemisah antar sel
    cbar_kws={'shrink': 0.8, 'label': 'Correlation'}  # colorbar lebih kecil +
)
plt.title("Correlation Heatmap - Network Traffic Data", fontsize=14, fontweigh
plt.show()

plt.figure(figsize=(12,8))
sns.heatmap(
    data2.corr(numeric_only=True),
    cmap="coolwarm",
    annot=True,
    fmt=".2f",
    linewidths=0.5,
    cbar_kws={'shrink': 0.8, 'label': 'Correlation'}
)
plt.title("Correlation Heatmap - Malware Detection Data", fontsize=14, fontwei
plt.show()
```



📊 Heatmap Korelasi

1. Range nilai korelasi

Nilai korelasi berada di rentang -1 sampai +1.

+1 → hubungan linear positif sempurna. Jika satu variabel naik, variabel lain juga naik dengan proporsi yang sama.

-1 → hubungan linear negatif sempurna. Jika satu variabel naik, variabel lain turun dengan proporsi yang sama.

0 → tidak ada hubungan linear yang jelas.

2. Warna pada heatmap

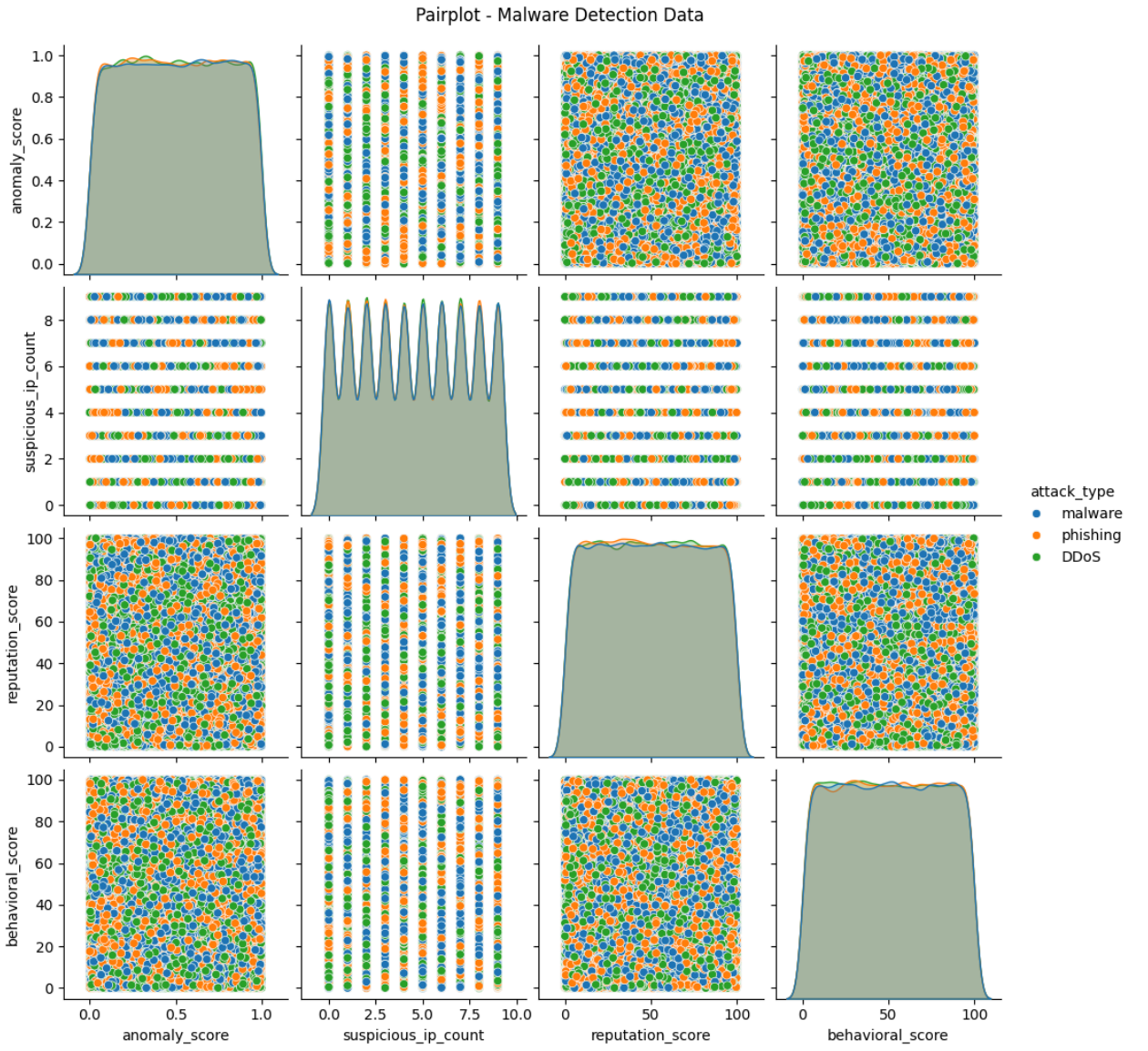
Merah pekat → korelasi positif tinggi.

Biru pekat → korelasi negatif tinggi.

Putih / warna pucat → korelasi sangat lemah atau hampir tidak ada hubungan.

In [48]:

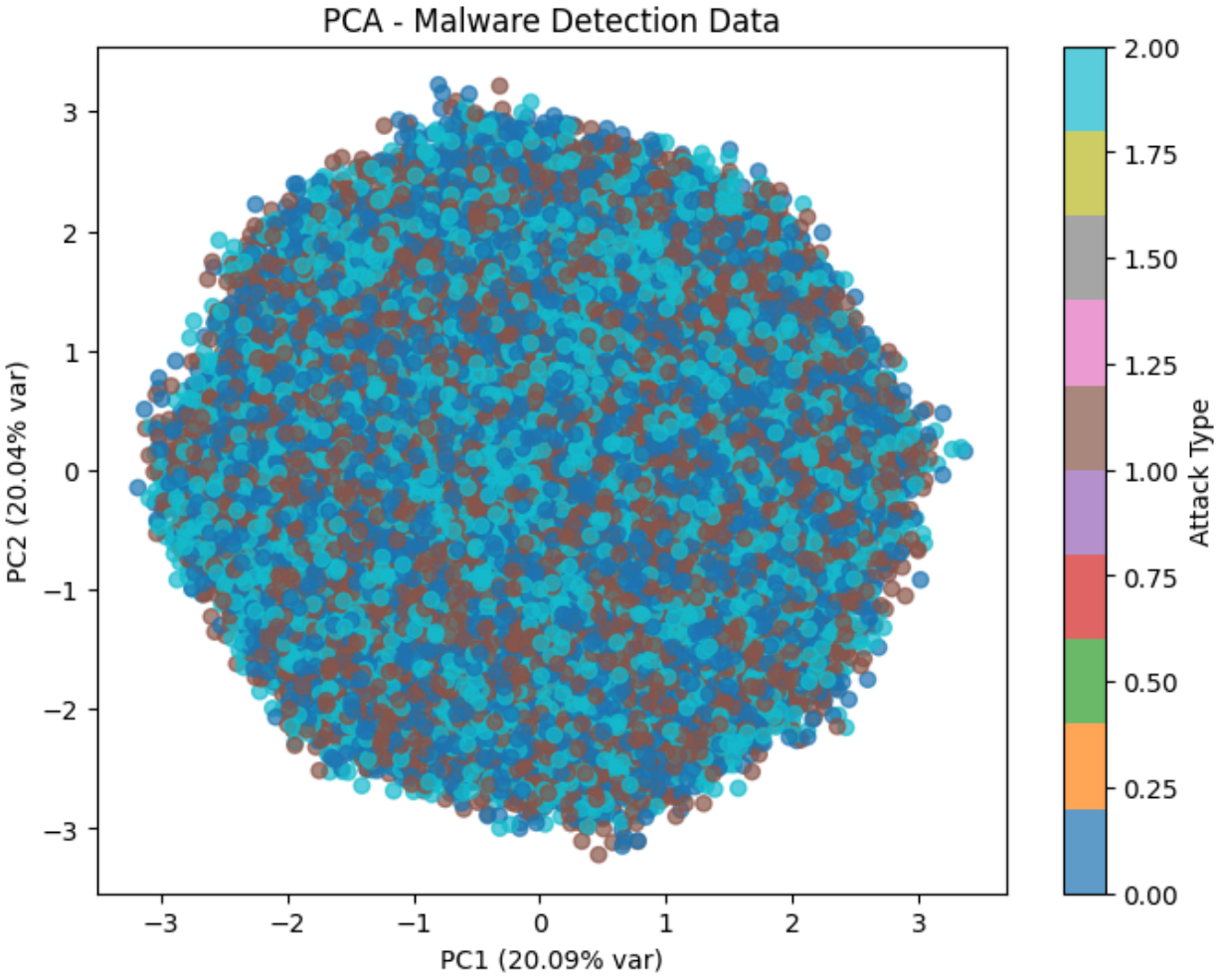
```
# Pairplot / scatter matrix untuk fitur penting
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# load dataset
data2 = pd.read_csv("/content/drive/MyDrive/STATPROB/DATASET/malware_detection

sns.pairplot(
    data2[[
        "anomaly_score",
        "suspicious_ip_count",
        "reputation_score",
        "behavioral_score",
        "attack_type"
    ]],
    diag_kind="kde",
    hue="attack_type"
)
plt.suptitle("Pairplot - Malware Detection Data", y=1.02)
plt.show()
```



In [49]:

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

#Standarisasi data numerik

features = ["anomaly_score", "suspicious_ip_count", "reputation_score", "behav
X = data2[features].dropna()
X_scaled = StandardScaler().fit_transform(X)

#PCA dengan 2 komponen

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

#Plot hasil PCA

plt.figure(figsize=(8,6))
plt.scatter(X_pca[:,0], X_pca[:,1], c=pd.factorize(data2.loc[X.index, "attack_
```

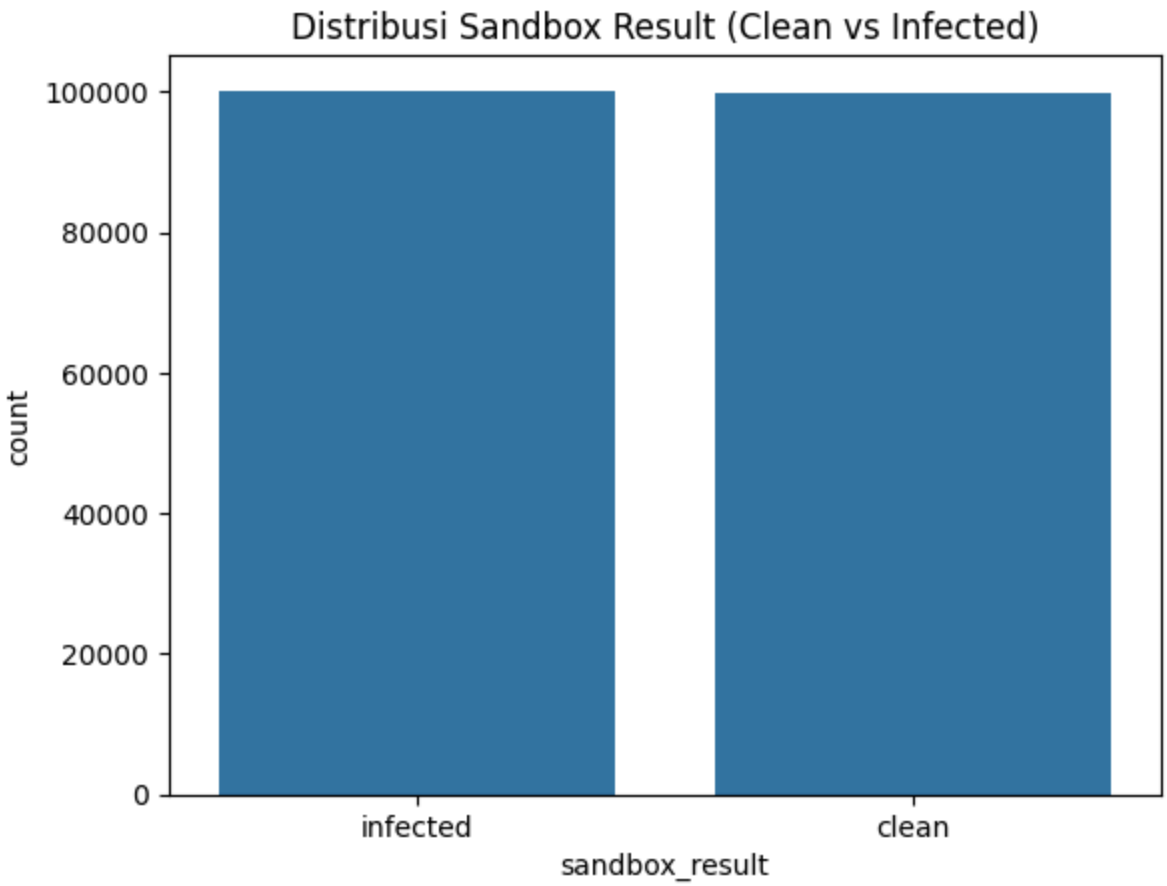
```
plt.xlabel("PC1 (%.2f%% var)" % (pca.explained_variance_ratio_[0]*100))
plt.ylabel("PC2 (%.2f%% var)" % (pca.explained_variance_ratio_[1]*100))
plt.title("PCA - Malware Detection Data")
plt.colorbar(label="Attack Type")
plt.show()
```

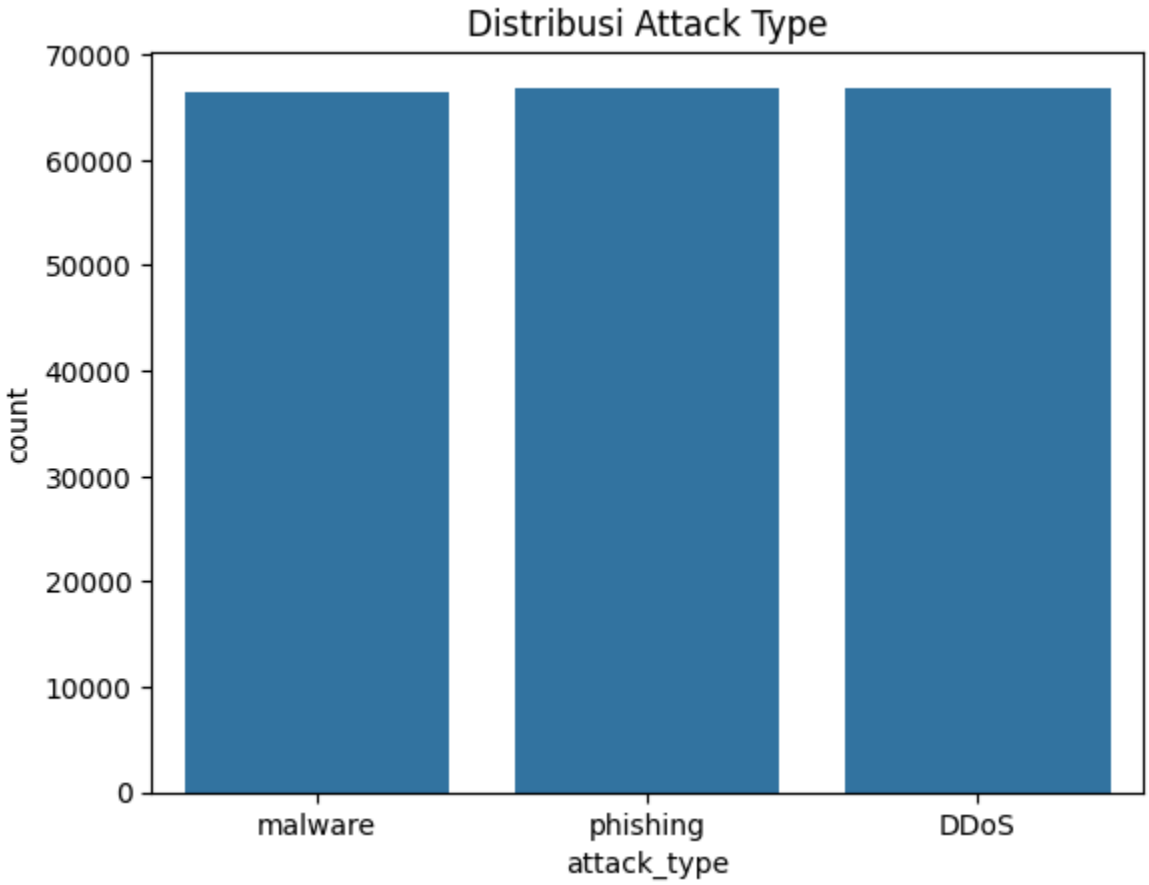


6. Insight & Kesimpulan Awal

- Pola menarik dari data
- Variabel paling berpengaruh
- Potensi masalah (multicollinearity, imbalance data, dsb.)
- Pertanyaan lanjutan untuk modeling

In [50]:
```
sns.countplot(x="sandbox_result", data=data2)
plt.title("Distribusi Sandbox Result (Clean vs Infected)")
plt.show()

sns.countplot(x="attack_type", data=data2)
plt.title("Distribusi Attack Type")
plt.show()
```


Distribusi Sandbox Result (Clean vs Infected)
100000
80000
60000
40000
20000
0
count
infected
clean
sandbox_result

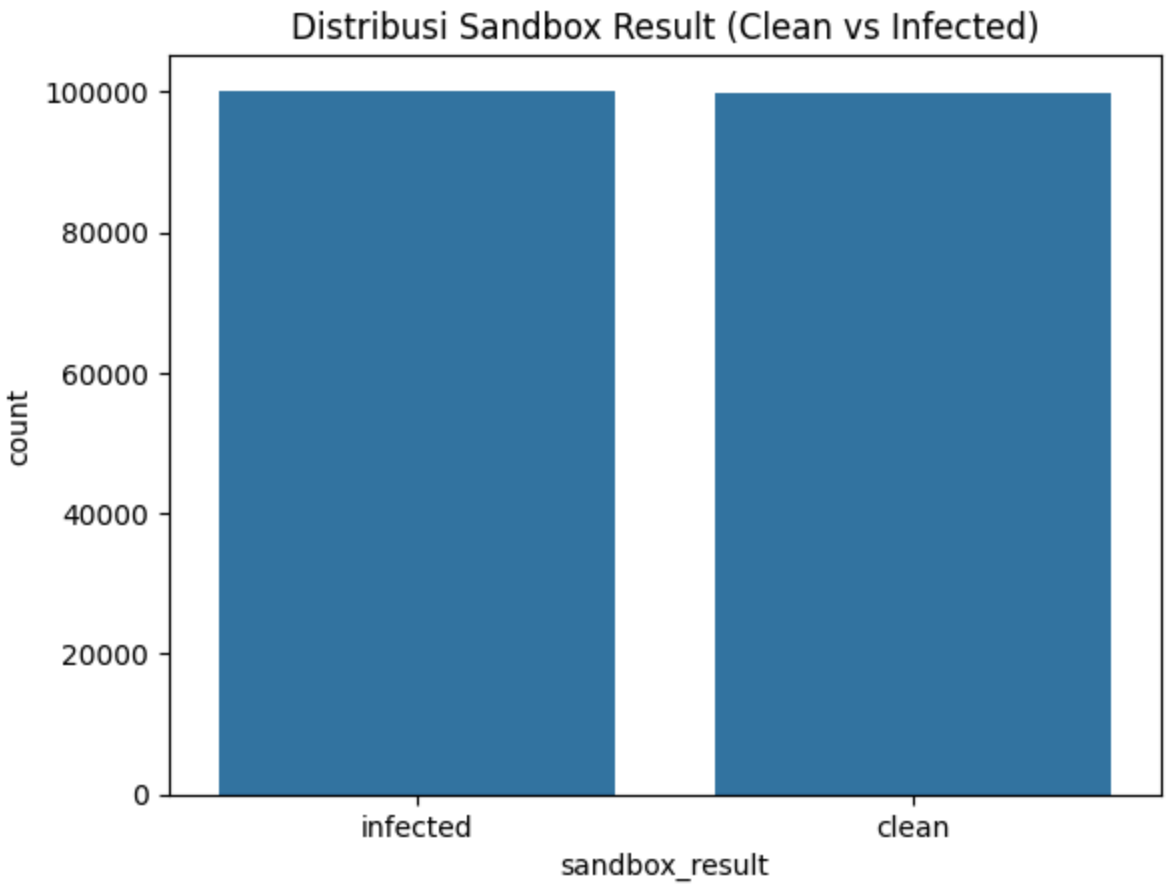

Distribusi Attack Type
70000
60000
50000
40000
30000
20000
10000
0
count
malware
phishing
DDoS
attack_type

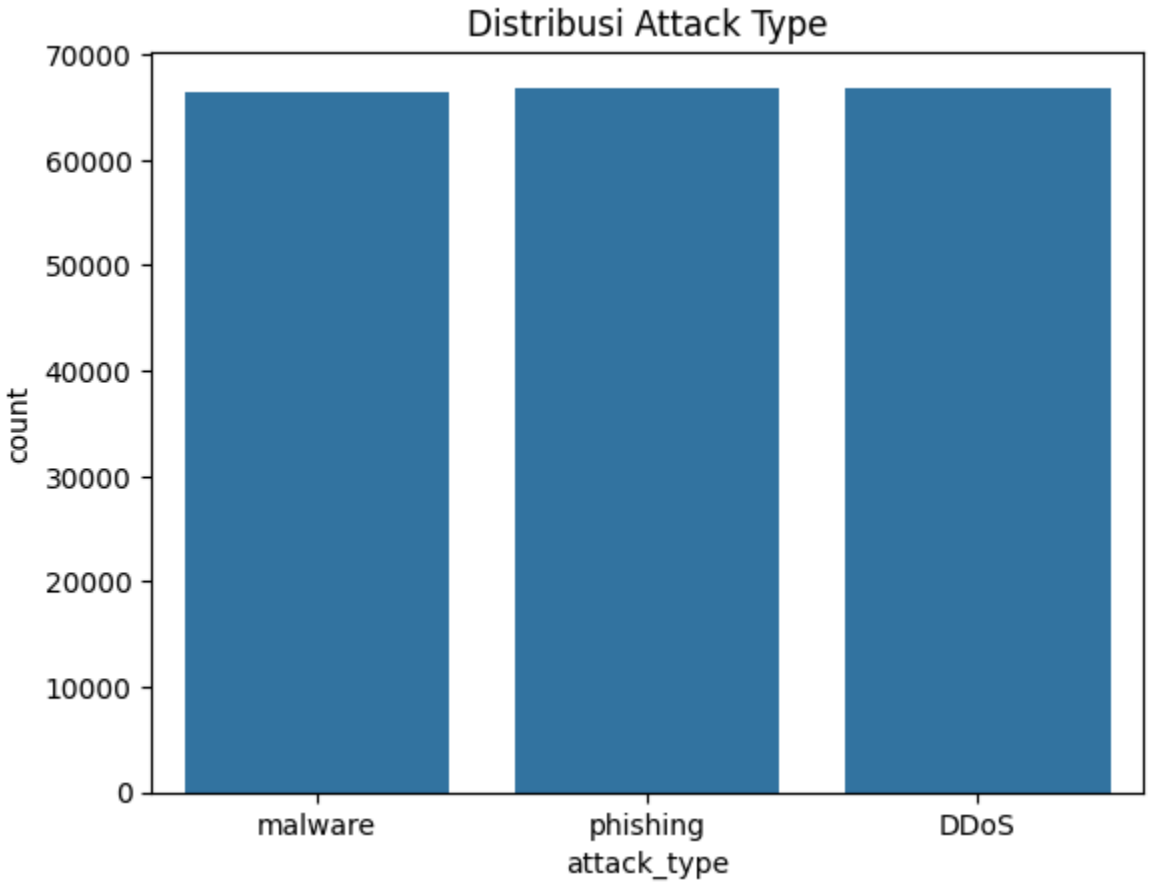
In [51]:
```
data1.describe(include="all")
data2.describe(include="all")
```

Out[51]:

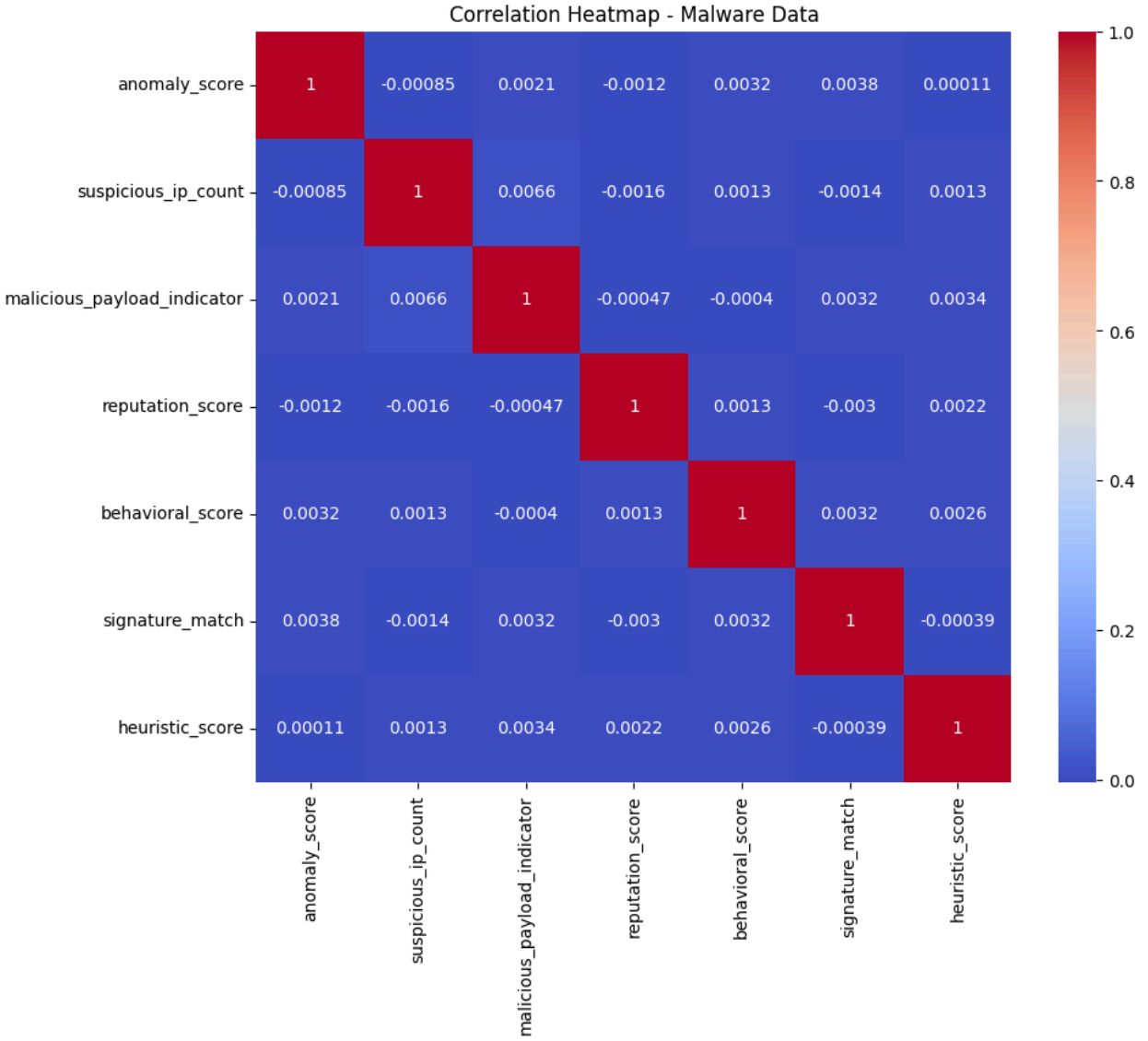
	anomaly_score	suspicious_ip_count	malicious_payload_indicator	reputa
count	200000.000000	200000.000000	200000.00000	200
unique	NaN	NaN	NaN	
top	NaN	NaN	NaN	
freq	NaN	NaN	NaN	
mean	0.500966	4.489850	0.49912	
std	0.288896	2.873703	0.50000	
min	0.000002	0.000000	0.00000	
25%	0.251379	2.000000	0.00000	
50%	0.500837	4.000000	0.00000	
75%	0.751856	7.000000	1.00000	
max	0.999994	9.000000	1.00000	

In [52]:
```
sns.countplot(x="sandbox_result", data=data2)
plt.title("Distribusi Sandbox Result (Clean vs Infected)")
plt.show()

sns.countplot(x="attack_type", data=data2)
plt.title("Distribusi Attack Type")
plt.show()
```


Distribusi Sandbox Result (Clean vs Infected)
100000
80000
60000
40000
20000
0
count
infected
clean
sandbox_result


Distribusi Attack Type
70000
60000
50000
40000
30000
20000
10000
0
count
malware
phishing
DDoS
attack_type

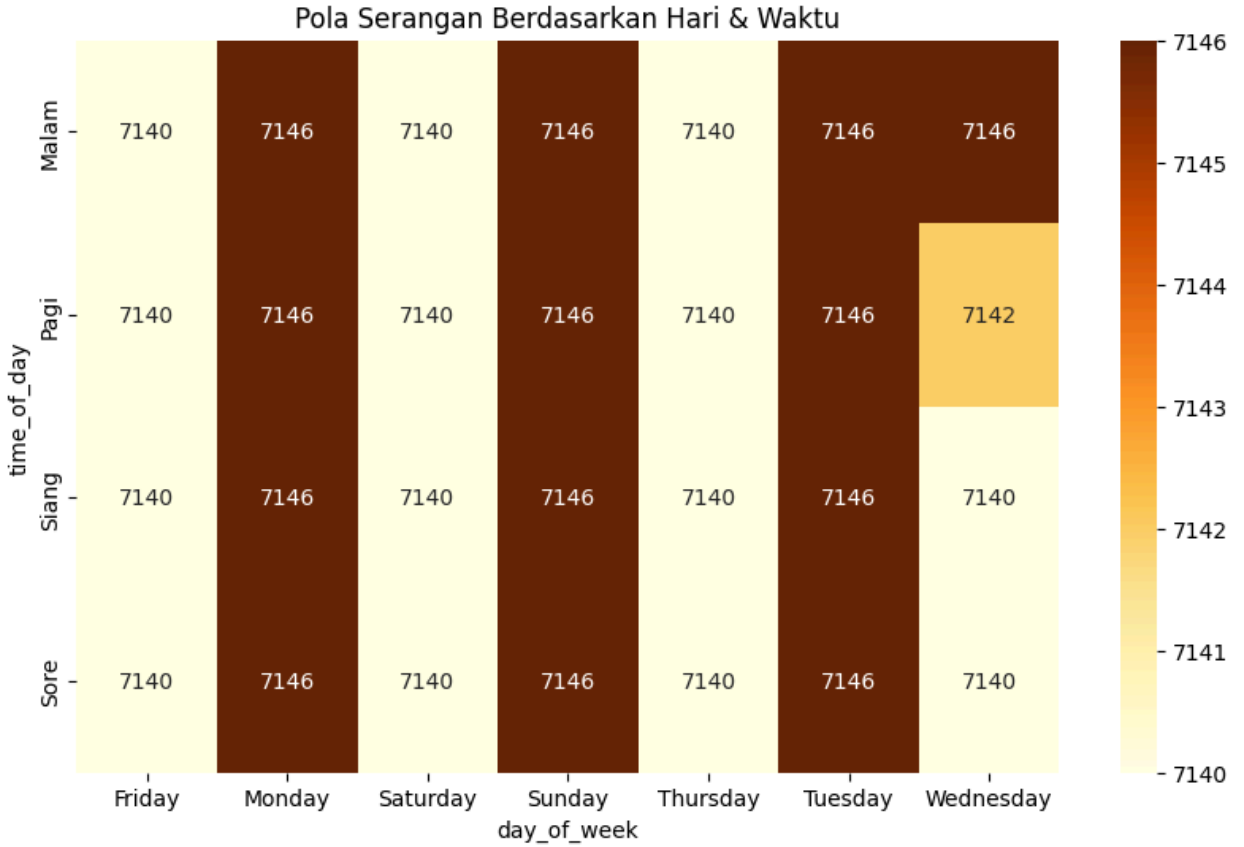
In [53]:
```
plt.figure(figsize=(10,8))
sns.heatmap(data2.corr(numeric_only=True), annot=True, cmap="coolwarm")
plt.title("Correlation Heatmap - Malware Data")
plt.show()
```



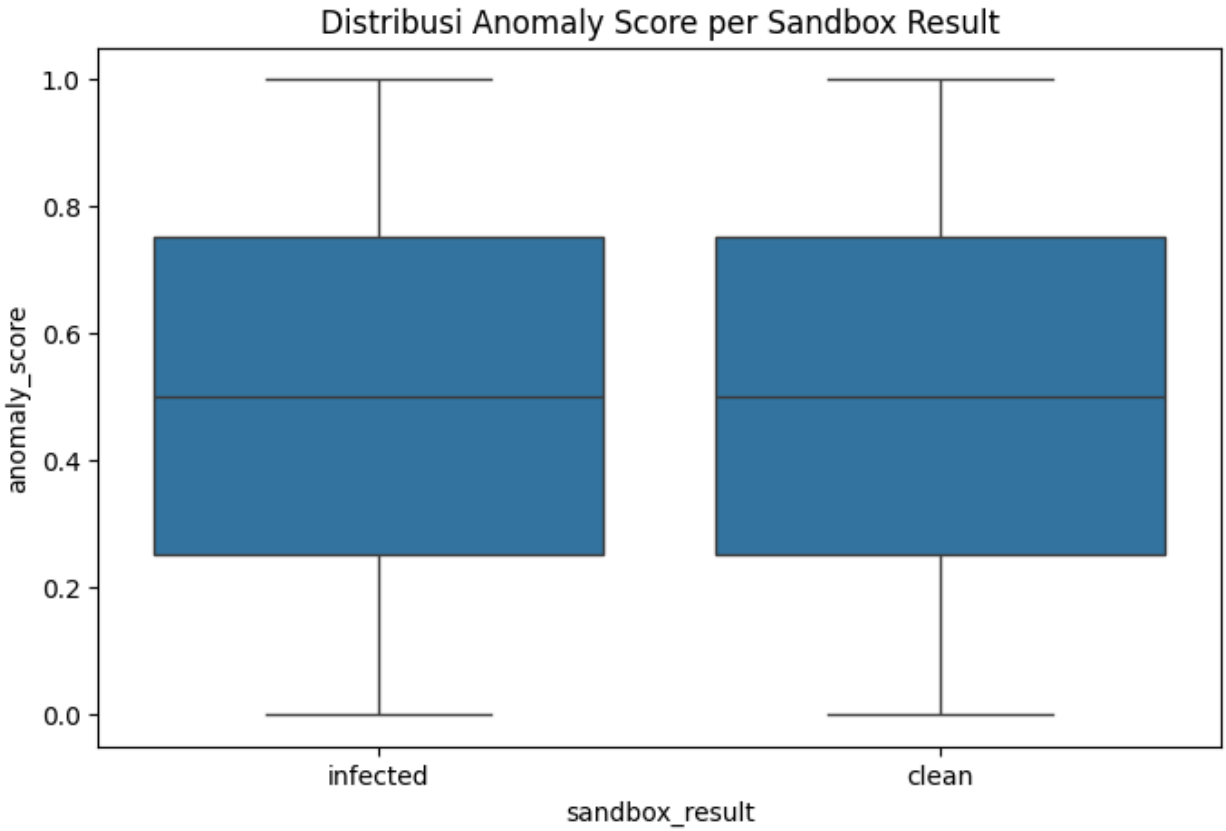
In [54]:
```
import numpy as np

# Bikin dummy timestamp (acak dalam range 7 hari)
data2['timestamp'] = pd.date_range(start="2023-01-01", periods=len(data2), fre

# Buat kolom hari & waktu dari timestamp
data2['day_of_week'] = data2['timestamp'].dt.day_name()
data2['time_of_day'] = pd.cut(
    data2['timestamp'].dt.hour,
    bins=[0, 6, 12, 18, 24],
    labels=['Malam', 'Pagi', 'Siang', 'Sore'],
    right=False
)
```

In [55]:
```
pivot = data2.pivot_table(
    index="time_of_day",
    columns="day_of_week",
    values="anomaly_score",
    aggfunc="count",
    fill_value=0
)

plt.figure(figsize=(10,6))
sns.heatmap(pivot, annot=True, fmt="d", cmap="YlOrBr")
plt.title("Pola Serangan Berdasarkan Hari & Waktu")
plt.show()
```

```
/tmp/ipython-input-2555502931.py:1: FutureWarning: The default value of observe
d=False is deprecated and will change to observed=True in a future version of p
andas. Specify observed=False to silence this warning and retain the current be
havior
  pivot = data2.pivot_table(
```



In [56]:
```
plt.figure(figsize=(8,5))
sns.boxplot(x="sandbox_result", y="anomaly_score", data=data2)
plt.title("Distribusi Anomaly Score per Sandbox Result")
plt.show()
```



Dalam data terlihat pola menarik bahwa trafik berdurasi panjang dengan byte besar lebih sering terkait aktivitas abnormal, sedangkan pada data malware skor anomaly dan heuristic tinggi hampir selalu berhubungan dengan status infected.

Beberapa variabel paling berpengaruh antara lain flow duration, packet length, dan bytes sent/received pada data trafik, serta anomaly score dan heuristic score pada data malware karena paling jelas membedakan serangan.

Ada pula potensi masalah seperti multicollinearity antar variabel numerik (misalnya antara durasi dan jumlah byte), serta kemungkinan imbalance data jika jumlah serangan lebih sedikit dibanding trafik normal.

Dari sini muncul pertanyaan lanjutan: fitur jaringan apa yang paling menentukan serangan? Bisakah skor deteksi diprediksi hanya dari trafik? Dan apakah pola serangan bervariasi menurut waktu, protokol, atau alamat IP?

#Uji Statistik In this project, the Spearman Rank Correlation was selected as the primary statistical test to examine the relationship between DNS Query Count (from the Network Traffic dataset) and Suspicious IP Count (from the Malware Detection dataset). The choice of Spearman correlation is justified by several important reasons:

Tujuan : Pengaruh Aktivitas DNS pada Trafik Jaringan terhadap Jumlah Aktivitas

Mencurigakan.

Pre-processing

0. **Python library that'll be used are:**

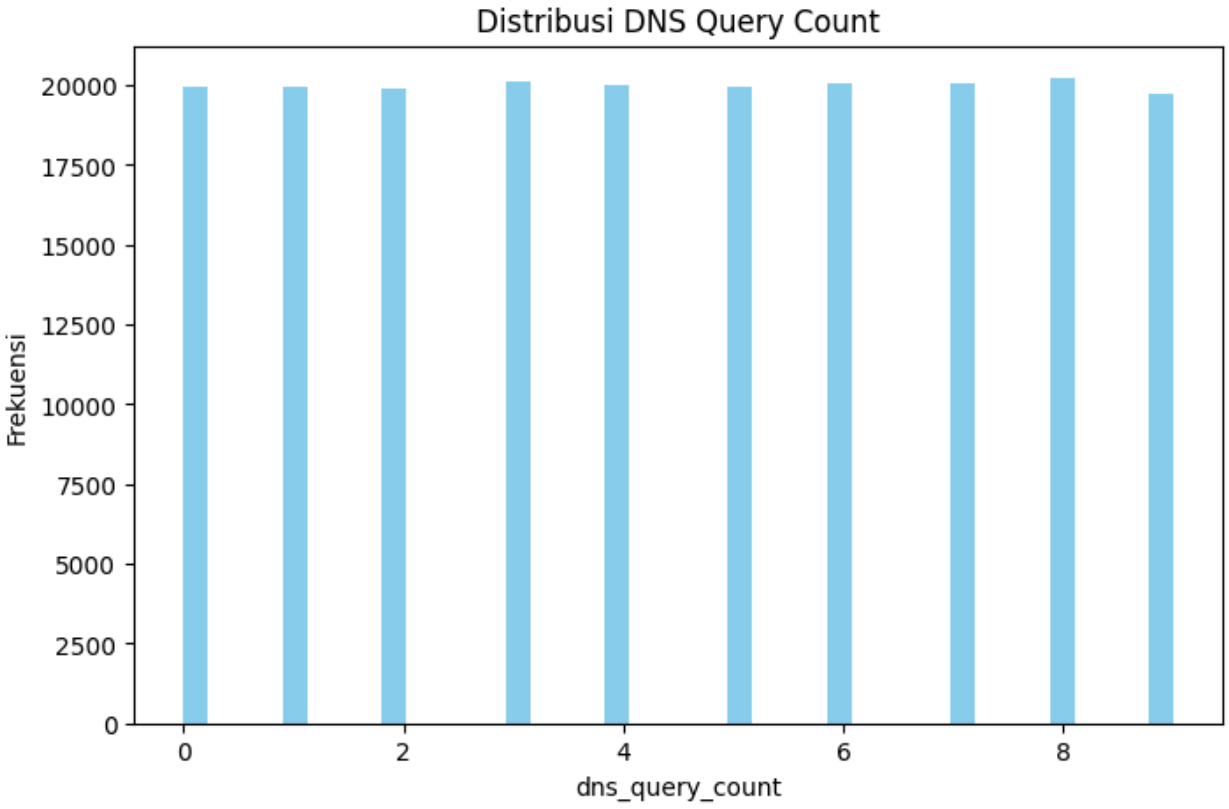
- SciPy (spearmanr) → for statistical hypothesis testing

In [57]:
```
from scipy.stats import spearmanr
```

1. **Dataset**

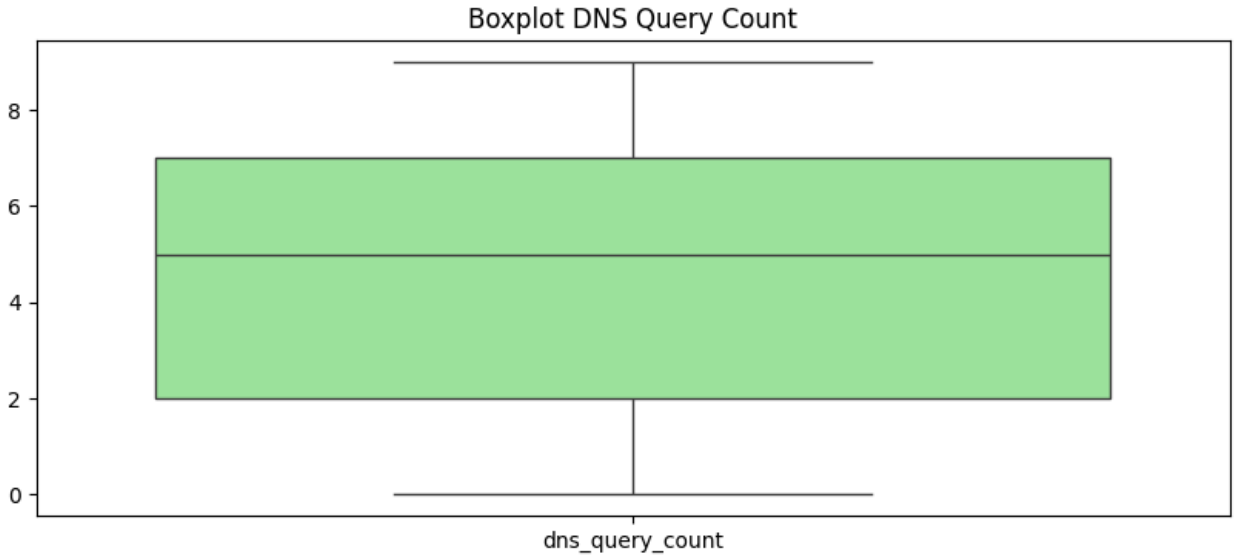
1.2. **Distribusi Variabel: dns_query_count**

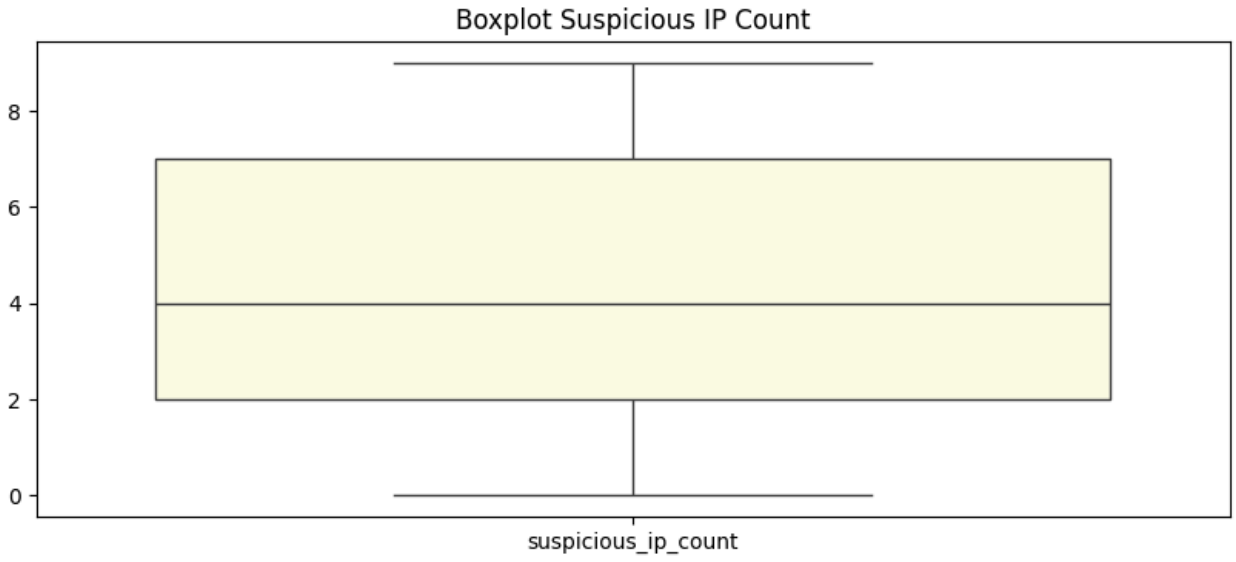
In [58]:
```
plt.figure(figsize=(8,5))
plt.hist(data1['dns_query_count'], bins=40, color='skyblue')
plt.title('Distribusi DNS Query Count')
plt.xlabel('dns_query_count')
plt.ylabel('Frekuensi')
plt.show()
```



1.3 **Boxplot untuk mendeteksi outlier**

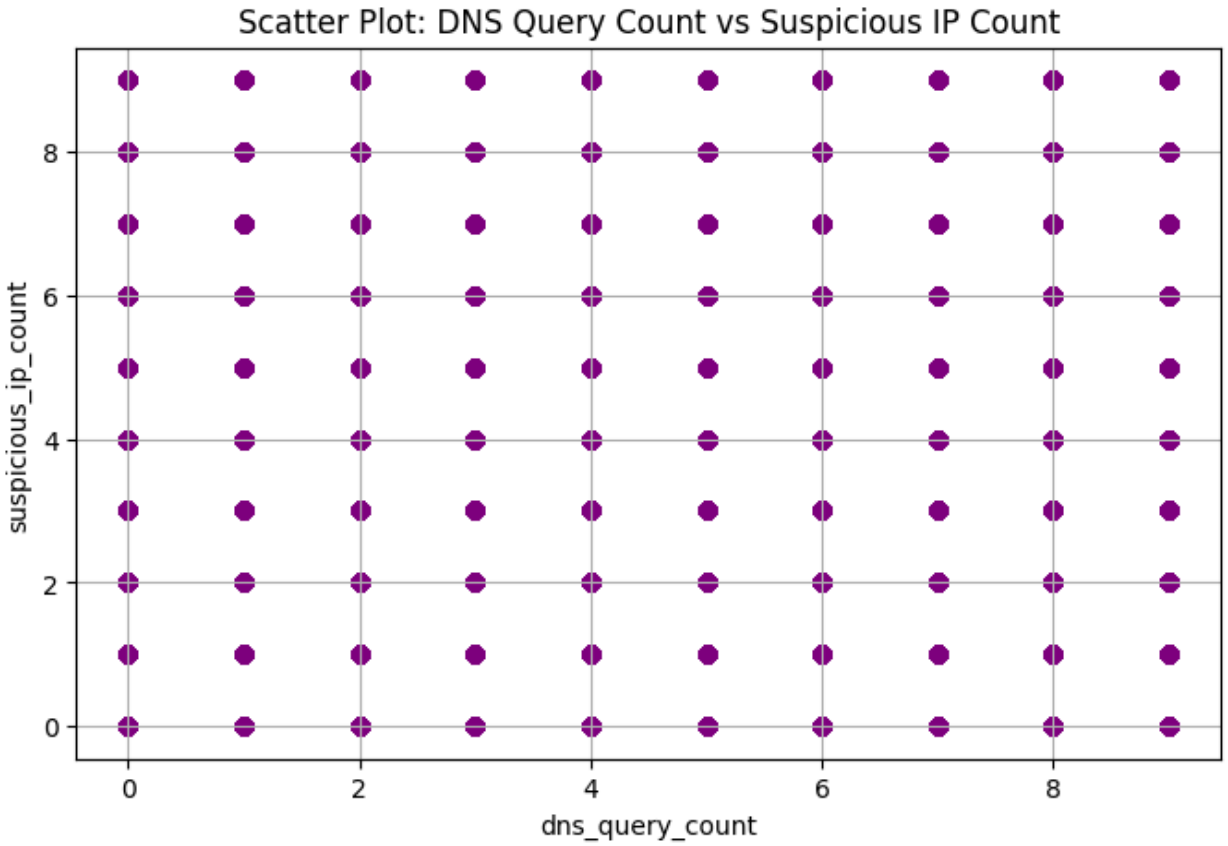
In [59]:
```
plt.figure(figsize=(10,4))
sns.boxplot(data=data1[['dns_query_count']], color='lightgreen')
plt.title('Boxplot DNS Query Count')
plt.show()

plt.figure(figsize=(10,4))
sns.boxplot(data=data2[['suspicious_ip_count']], color='lightyellow')
plt.title('Boxplot Suspicious IP Count')
plt.show()
```





In [60]:
```
n = min(len(data1), len(data2))
dns = data1['dns_query_count'].iloc[:n]
sip = data2['suspicious_ip_count'].iloc[:n]

plt.figure(figsize=(8,5))
```

```
plt.scatter(dns, sip, alpha=0.4, color='purple')
plt.title('Scatter Plot: DNS Query Count vs Suspicious IP Count')
plt.xlabel('dns_query_count')
plt.ylabel('suspicious_ip_count')
plt.grid(True)
plt.show()
```



2. **Tema Besar Project (Judul)**

Pengaruh Aktivitas DNS pada Trafik Jaringan terhadap Jumlah Aktivitas Mencurigakan.

3. **Pertanyaan Statistik**

Apakah rata-rata jumlah IP mencurigakan (suspicious_ip_count) lebih tinggi pada trafik dengan aktivitas DNS yang tinggi dibandingkan trafik dengan aktivitas DNS yang rendah (berdasarkan kategori dns_query_count)?

3.2 **Hipotesis Penelitian**

H0 (Hipotesis Nol):
Tidak terdapat hubungan antara aktivitas DNS (`dns_query_count`) dan jumlah aktivitas mencurigakan (`suspicious_ip_count`).

H1 (Hipotesis Alternatif):
Terdapat hubungan antara aktivitas DNS (`dns_query_count`) dan aktivitas mencurigakan (`suspicious_ip_count`).

3.3 **Jenis Uji Statistik**

Jenis Uji Statistik yang digunakan adalah uji Korelasi Spearman. Uji ini dipilih karena kedua variabel (dns_query_count dan suspicious_ip_count) merupakan data numerik dengan distribusi tidak normal dan pola hubungan yang tidak linier. Uji Spearman cocok digunakan untuk menentukan apakah terdapat hubungan signifikan antara dua variabel tanpa asumsi normalitas.

3.4 **Uji statistik**

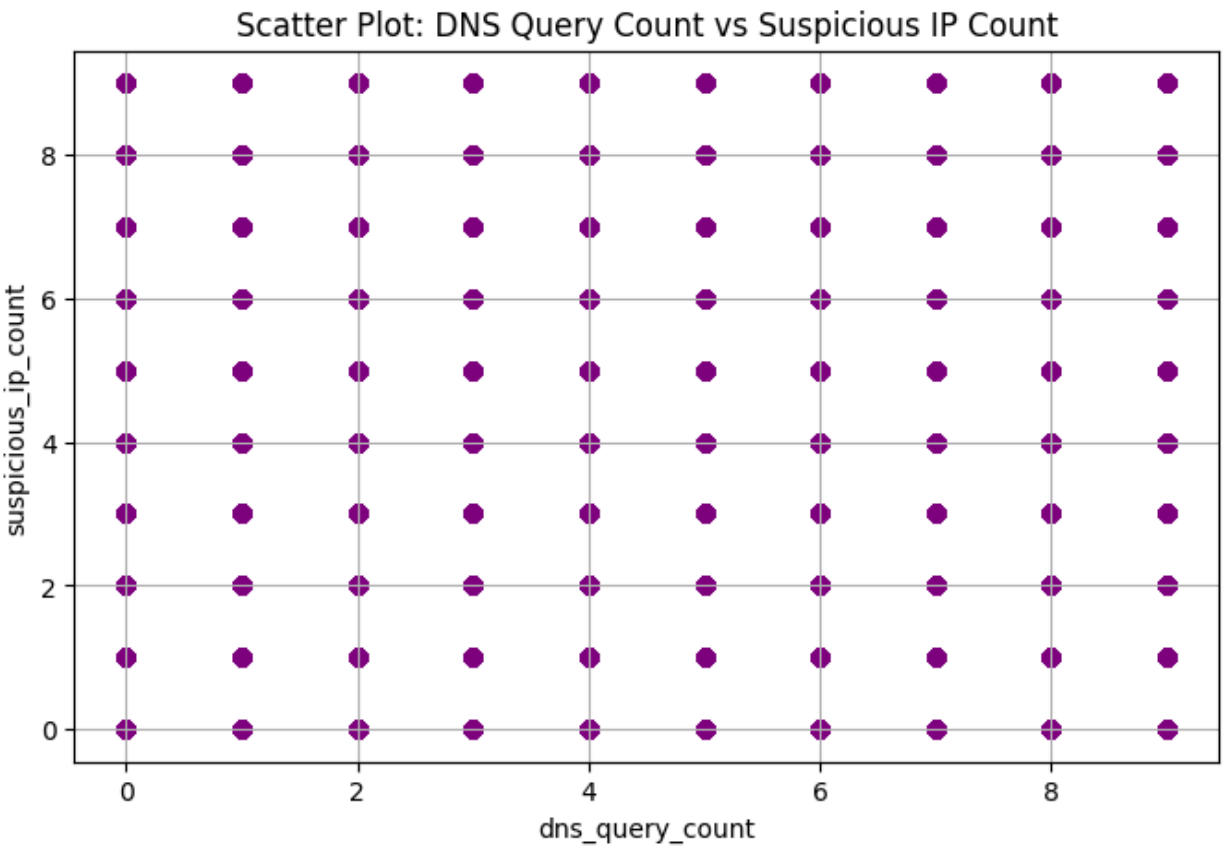
In [61]:
```
rho, pval = spearmanr(dns, sip)

print("=== Spearman Correlation Result ===")
print(f"Koefisien Spearman (rho): {rho}")
print(f"P-value: {pval}")

if pval < 0.05:
    print("\nKesimpulan: Terdapat hubungan yang signifikan.")
else:
    print("\nKesimpulan: Tidak terdapat hubungan signifikan.")
```

```
=== Spearman Correlation Result ===
Koefisien Spearman (rho): -0.0039376478275888775
P-value: 0.07824424876961163

Kesimpulan: Tidak terdapat hubungan signifikan.
```

3.6 **Analisis Tambahan**

In [62]:
```
plt.figure(figsize=(8,5))
plt.scatter(dns, sip, alpha=0.4, color='purple')
plt.title('Scatter Plot: DNS Query Count vs Suspicious IP Count')
plt.xlabel('dns_query_count')
plt.ylabel('suspicious_ip_count')
plt.grid(True)
```

```
plt.show()
```



Berdasarkan grafik dan distribusi data :

- dns_query_count memiliki variasi alami dari aktivitas normal jaringan.
- suspicious_ip_count muncul dalam pola yang lebih acak, kemungkinan besar berkaitan dengan scanning, probing, atau trafik berbahaya lain.
- Pola antar variabel tidak menunjukkan tren naik/turun yang konsisten, selaras dengan hasil korelasi Spearman.

3.6.1 **Implikasi Hasil Analisis dalam Konteks Cybersecurity**

1. Aktivitas DNS *tidak dapat* dijadikan indikator utama deteksi serangan.
2. Banyak permintaan DNS berasal dari aktivitas normal (update, browsing, API).
3. Aktivitas mencurigakan lebih terkait pada scanning, brute force, dan C2 traffic.
4. Sistem keamanan sebaiknya memprioritaskan indikator seperti:
    - anomaly_score
    - suspicious signature
    - traffic_volume

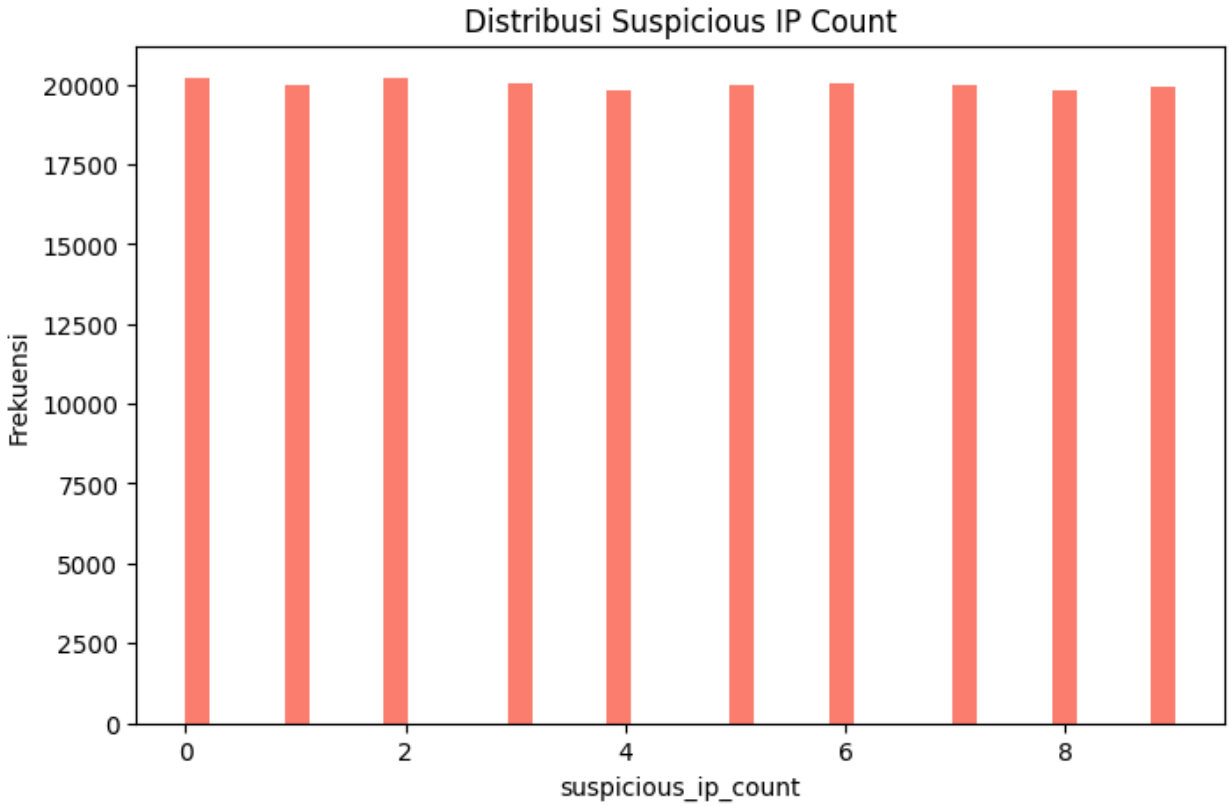
    - flow_duration
5. Hasil analisis membantu mengurangi false alarm ketika lonjakan DNS terjadi.

3.6.2 **Rekomendasi Keamanan**

1. Jangan bergantung hanya pada indikator DNS untuk deteksi ancaman.
2. Gunakan multi-indikator seperti traffic anomaly, suspicious IP reputation, atau behavioral score.
3. Implementasikan pemantauan pada protokol HTTP, TCP, dan port scanning.
4. Lakukan analisis mendalam terhadap pola traffic berbasis volume dan durasi.
5. Kombinasikan hasil ini dengan machine learning atau rule-based detection.

4. **Distribusi Variabel: suspicious_ip_count**

In [63]:
```
plt.figure(figsize=(8,5))
plt.hist(data2['suspicious_ip_count'], bins=40, color='salmon')
plt.title('Distribusi Suspicious IP Count')
plt.xlabel('suspicious_ip_count')
plt.ylabel('Frekuensi')
plt.show()
```



5. **Kesimpulan**

Berdasarkan uji korelasi Spearman diperoleh:

- rho ≈ -0.0039 (sangat lemah)
- p-value > 0.05 (tidak signifikan)

Maka:

- **H0 gagal ditolak**, artinya *tidak terdapat hubungan signifikan* antara aktivitas DNS dan aktivitas mencurigakan.
- DNS query bukan indikator efektif untuk mendeteksi ancaman malware pada dataset ini.