Traffic Analysis Using the CTU-13 Dataset

Botnet Behavior
Analysis and Detection

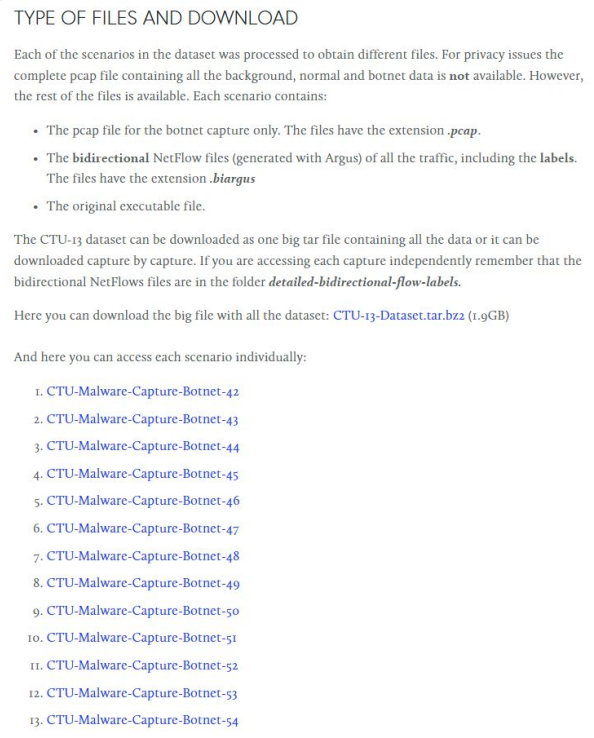
Overview

The CTU-13 dataset consists of multiple PCAP (Packet Capture) files, each corresponding to various botnets used

- The focus is on detection, and prevention, of botnet behavior based on various traffic features found in realistic traffic scenarios

Research aimed to:

- Identify traffic patterns in botnet activity
- Differentiate between malicious and normal traffic
- Examine the differences between various botnets



TYPE OF FILES AND DOWNLOAD

Each of the scenarios in the dataset was processed to obtain different files. For privacy issues the complete pcap file containing all the background, normal and botnet data is **not** available. However, the rest of the files is available. Each scenario contains:

- The pcap file for the botnet capture only. The files have the extension ***.pcap***.
- The **bidirectional** NetFlow files (generated with Argus) of all the traffic, including the **labels**. The files have the extension ***.biargus***
- The original executable file.

The CTU-13 dataset can be downloaded as one big tar file containing all the data or it can be downloaded capture by capture. If you are accessing each capture independently remember that the bidirectional NetFlows files are in the folder ***detailed-bidirectional-flow-labels***.

Here you can download the big file with all the dataset: CTU-13-Dataset.tar.bz2 (1.9GB)

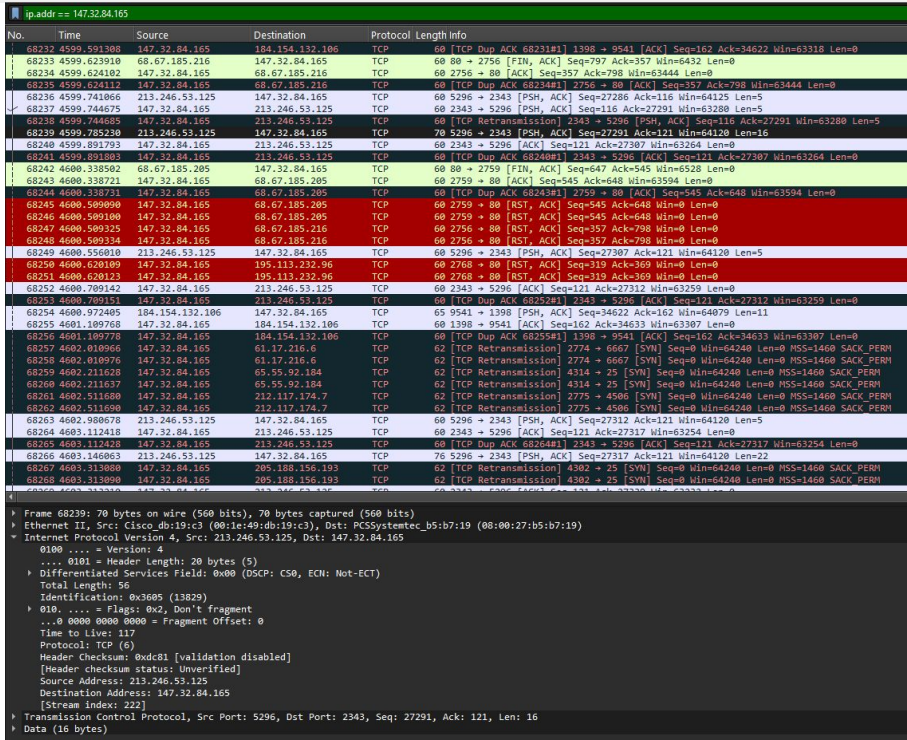
And here you can access each scenario individually:

1. CTU-Malware-Capture-Botnet-42
2. CTU-Malware-Capture-Botnet-43
3. CTU-Malware-Capture-Botnet-44
4. CTU-Malware-Capture-Botnet-45
5. CTU-Malware-Capture-Botnet-46
6. CTU-Malware-Capture-Botnet-47
7. CTU-Malware-Capture-Botnet-48
8. CTU-Malware-Capture-Botnet-49
9. CTU-Malware-Capture-Botnet-50
10. CTU-Malware-Capture-Botnet-51
11. CTU-Malware-Capture-Botnet-52
12. CTU-Malware-Capture-Botnet-53
13. CTU-Malware-Capture-Botnet-54

Dataset Features

The CTU-13 dataset includes traffic from 13 botnet scenarios and normal traffic.
Packet-level details:

- Timestamps
- Source and Destination IP addresses
- Ethernet Source and Destination addresses
- Communication protocols
- Packet sizes
- Network flags and metadata



Research Questions

What patterns can be identified in botnet traffic?

Explore unusual packet sizes, frequent connections, and protocol usage

How can we visualize differences between safe and malicious traffic?

Use visual techniques (histograms, time-series graphs, scatter plots, ect) to highlight differences

What data-driven differences exist between different botnet families?

Identify unique characteristics in packet size, traffic frequency, and protocol use

Dataset Overview

Index of /publicDatasets/CTU-Malware-Capture-Botnet-42

Name	Last modified	Size	Description
Parent Directory		-	
Neris.exe.zip	2015-12-16 10:28	46K	
README.html	2017-05-11 13:47	5.7K	
README.md	2017-05-21 18:42	5.0K	
botnet-capture-20110810-neris.html	2015-05-14 11:58	9.3M	
botnet-capture-20110810-neris.json	2015-05-14 11:58	14M	
botnet-capture-20110810-neris.pcap	2011-08-11 09:31	56M	
bro/	2017-04-14 22:33	-	
capture20110810.binetflow.2format	2017-05-08 16:42	664M	
capture20110810.truncated.pcap.bz2	2015-07-17 19:52	1.2G	
detailed-bidirectional-flow-labels/	2015-05-14 11:50	-	
ralabel-flowfilter.conf.generic	2014-07-18 10:09	79K	

CTU-Malware-Capture-Botnet-42 or Scenario 1 in the CTU-13 dataset.

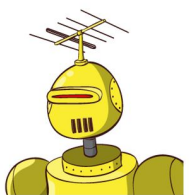
Description

- Probable Name: Neris
- MD5: bf08e6b02e00d2bc6dd493e93e69872f
- SHA1: 5c2ba68d78471ff02adcdab12b2f82db8efe2104
- SHA256: 527da5fd4e501765cdd1bccb2f7c5ac76c0b22dfaf7c24e914df4e1cb8029d71
- Password of zip file: infected
- Duration: 6.15 hours
- Complete Pcap size: 52GB
- Botnet Pcap size: 56MB

- NetFlow size: 1GB

- VirusTotal
- HybridAnalysis

- RobotHash



Botnet	Packets Captured	Collection Date
Neris	98,936	Aug 10, 2011
Rbot	99,989	Aug 12, 2011
Virut	45,809	Aug 15, 2011
DonBot	24,741	Aug 16, 2011
Sogou	20,639	Aug 16, 2011
Murlo	85,498	Aug 17, 2011
NSIS.ay	97,470	Aug 17, 2011
Combined	1,000,000+	Aug 10, 2011 - Aug 17, 2011

Analysis of 7 Botnets

[Neris, Rbot, Virut, DonBot, Sogou, Murlo, NSIS.ay, and a Combined Dataset]

Methodology

- TShark was used to extract data from the .pcap files, converting to CSV for analysis
- Utilised the web based analysis tool Jupyter Notebook
- Data was processed using Python libraries (numpy, pandas, matplotlib, scipy)

Data Cleaning:

- Handle missing values in protocols, packet sizes, and IP addresses

Statistical Analysis:

- Descriptive statistics for traffic patterns (mean, min, max, standard deviation, distribution)
- Outlier detection using the IQR method to identify anomalous traffic
- Temporal analysis to identify peak botnet activity by hour and day
- Protocol analysis for understanding botnet preferences

Data Extraction and Processing

The botnet .pcap conversions were done using TShark to easily extract the relevant data I needed for this analysis.
Fields being analyzed are as follows: (frame.time, ip.src, ip.dst, ip.len, ip.proto, tcp.srcport, tcp.dstport, eth.src, eth.dst, udp.srcport, udp.dstport, udp.length, icmp.type, icmp.code, icmp.seq, dns.qry.name, dns.qry.type, and dns.a) with the data using a capture packet amount of -c 100000 packets.

```
C:\Users\Harry\Downloads>tshark -r botnet-capture-20110810-neris.pcap -T fields -e frame.time -e ip.src -e ip.dst -e ip.
len -e ip.proto -e tcp.srcport -e tcp.dstport -e eth.src -e eth.dst -e udp.srcport -e udp.dstport -e udp.length -e icmp.
type -e icmp.code -e icmp.seq -e dns.qry.name -e dns.qry.type -e dns.a -c 100000 -E separator=, > extracted_data.csv
```

here the (all_botnet_captures.pcap) file contained a grouping of all pcap captures which all analyzed the same fields, however, this capture only received part of the results compared to the other captures.
This might have been due to the large packet size as the tshark functions did not limit the total packets during the conversion through -c <number>. But the results could have also derived from the merging process used.

```
C:\Users\Harry\Downloads>mergecap -w all_botnet_captures.pcap botnet-capture-20110810-neris.pcap botnet-capture-20110811-neris.pcap botnet-capture
-20110812-rbot.pcap botnet-capture-20110815-fast-flux.pcap botnet-capture-20110816-donbot.pcap botnet-capture-20110816-sogou.pcap botnet-capture-2
0110815-rbot-dos.pcap botnet-capture-20110816-qvod.pcap botnet-capture-20110815-fast-flux-2.pcap botnet-capture-20110819-bot.pcap botnet-capture-2
0110817-bot.pcap
```

mergecap assumes that all packet captures are already correctly ordered, which was to be expected.
However the conversion of the captures left some errors while being applied to Jupyter Notebook.

```
C:\Users\Harry\Downloads>tshark -r all_botnet_captures.pcap -T fields -e frame.time -e ip.src -e ip.dst -e ip.len -e ip.
proto -e tcp.srcport -e tcp.dstport -e eth.src -e eth.dst -e udp.srcport -e udp.dstport -e udp.length -e icmp.type -e ic
mp.code -e icmp.seq -e dns.qry.name -e dns.qry.type -e dns.a -E separator=, > all_extracted_data.csv
```

[57]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import re
from datetime import datetime

# Define constants and mappings
PROTOCOL_MAP = {
    1: 'ICMP',  # ICMP (Internet Control Message Protocol)
    6: 'TCP',   # TCP (Transmission Control Protocol)
    17: 'UDP',  # UDP (User Datagram Protocol)
    89: 'ICMPv6',  # ICMPv6 (Internet Control Message Protocol v6)
    2: 'IGMP',  # IGMP (Internet Group Management Protocol)
    41: 'IPv6',  # IPv6 (Internet Protocol version 6)
    50: 'ESP',  # ESP (Encapsulating Security Payload)
    51: 'AH',   # AH (Authentication Header)
    58: 'ICMPv6',  # ICMPv6 (Internet Control Message Protocol v6)
    0: 'Unknown',  # Unknown Protocol
    3: 'ARP',   # ARP (Address Resolution Protocol)
    132: 'SCTP',  # SCTP (Stream Control Transmission Protocol)
    33: 'DCCP',  # DCCP (Datagram Congestion Control Protocol)
    115: 'L2TP',  # L2TP (Layer 2 Tunneling Protocol)
    47: 'GRE',   # GRE (Generic Routing Encapsulation)
    88: 'EIGRP',  # EIGRP (Enhanced Interior Gateway Routing Protocol)
    89: 'OSPF',  # OSPF (Open Shortest Path First)
    39: 'LDP',   # LDP (Label Distribution Protocol)
    25: 'SMTP',  # SMTP (Simple Mail Transfer Protocol)
    53: 'DNS',   # DNS (Domain Name System)
    80: 'HTTP',  # HTTP (Hypertext Transfer Protocol)
    443: 'HTTPS',  # HTTPS (Hypertext Transfer Protocol Secure)
    21: 'FTP',   # FTP (File Transfer Protocol)
    23: 'Telnet',  # Telnet (Telecommunication Network)
    22: 'SSH',   # SSH (Secure Shell)
    123: 'NTP',   # NTP (Network Time Protocol)
}

PORT_PROTOCOL_MAP = {
    80: 'HTTP',           # HTTP (Hypertext Transfer Protocol)
    443: 'HTTPS',         # HTTPS (Hypertext Transfer Protocol Secure)
    21: 'FTP',            # FTP (File Transfer Protocol)
    22: 'SSH',            # SSH (Secure Shell)
    23: 'Telnet',         # Telnet (Telecommunication Network)
    25: 'SMTP',           # SMTP (Simple Mail Transfer Protocol)
    53: 'DNS',            # DNS (Domain Name System)
    110: 'POP3',          # POP3 (Post Office Protocol v3)
    143: 'IMAP',          # IMAP (Internet Message Access Protocol)
    3306: 'MySQL',        # MySQL (MySQL Database Service)
    8080: 'HTTP-Alt',     # HTTP Alternate (HTTP Alternative)
    8081: 'HTTP-Alt-2',   # HTTP Alternate 2 (commonly used for dev/test environments)
    67: 'DHCP Server',    # DHCP (Dynamic Host Configuration Protocol - Server)
    68: 'DHCP Client',    # DHCP (Dynamic Host Configuration Protocol - Client)
    69: 'TFTP',           # TFTP (Trivial File Transfer Protocol)
    161: 'SNMP',          # SNMP (Simple Network Management Protocol)
    162: 'SNMP Trap',     # SNMP (SNMP Trap for alerts)
    514: 'Syslog',        # Syslog (System Log Protocol)
    1433: 'MS SQL',       # MS SQL Server (Microsoft SQL Server)
    1434: 'MS SQL',       # MS SQL Server (Microsoft SQL Server - Browser Service)
    119: 'NNTP',          # NNTP (Network News Transfer Protocol)
    3389: 'RDP',          # RDP (Remote Desktop Protocol)
    465: 'SMTPS',         # SMTPS (SMTP Secure over SSL/TLS)
    993: 'IMAPS',         # IMAPS (IMAP Secure over SSL/TLS)
    995: 'POP3S',         # POP3S (POP3 Secure over SSL/TLS)
    1080: 'SOCKS Proxy',  # SOCKS Proxy
    2525: 'SMTP-Alt',     # SMTP Alternate (commonly used in email servers)
    5432: 'PostgreSQL',   # PostgreSQL (PostgreSQL Database Service)
    6379: 'Redis',        # Redis (Redis Key-Value Store)
    5900: 'VNC',          # VNC (Virtual Network Computing)
    6660: 'IRC',          # IRC (Internet Relay Chat)
    6661: 'IRC',          # IRC (Internet Relay Chat)
    6662: 'IRC',          # IRC (Internet Relay Chat)
    6663: 'IRC',          # IRC (Internet Relay Chat)
    6664: 'IRC',          # IRC (Internet Relay Chat)
    6665: 'IRC',          # IRC (Internet Relay Chat)
    6666: 'IRC',          # IRC (Internet Relay Chat)
    6667: 'IRC',          # IRC (Internet Relay Chat)
    6668: 'IRC',          # IRC (Internet Relay Chat)
    6669: 'IRC',          # IRC (Internet Relay Chat)
}

# Read the CSV file
def load_data(file_path):
    try:
        df = pd.read_csv(file_path, sep=',', engine='python', skip_blank_lines=True, on_bad_lines='skip',
                         names=['Timestamp', 'Source IP', 'Destination IP', 'Total Length', 'Protocol',
                                'TCP Source Port', 'TCP Destination Port', 'Ethernet Source', 'Ethernet Destination',
                                'UDP Source Port', 'UDP Destination Port', 'UDP Length', 'ICMP Type', 'ICMP Code',
                                'ICMP Seq', 'DNS Query Name', 'DNS Query Type', 'DNS A'])
        print("File loaded successfully.")
        return df
    except Exception as e:
        print(f"Error loading file: {e}")
        return None

# Clean and parse the 'Timestamp' column
def clean_and_parse_timestamp(timestamp):
    if isinstance(timestamp, str):
        if re.match(r'^\w{3} \d{2}$', timestamp):  # Matches 'Aug 10'
            timestamp = '2011 ' + timestamp + ' 00:00:00'  # Add default time if missing
        timestamp = re.sub(r"\s+[A-Za-z\s]+$", "", timestamp)  # Remove timezone info

        try:
            return pd.to_datetime(timestamp, errors='coerce')
        except Exception as e:
            print(f"Error parsing timestamp: {timestamp} -> {e}")
            return pd.NaT
    return pd.NaT

# Feature engineering: Adding useful time-based features
def add_time_features(df):
    df['Hour'] = df['Timestamp'].dt.hour
    df['DayOfWeek'] = df['Timestamp'].dt.dayofweek
    df['Weekday'] = df['Timestamp'].dt.weekday
    return df

# Clean 'Total Length' and fill missing values
def clean_total_length(df):
    df['Total Length'] = pd.to_numeric(df['Total Length'], errors='coerce')
    df['Total Length'] = df['Total Length'].fillna(df['Total Length'].median())
    return df

# Clean 'Source IP' and 'Destination IP'
def clean_ip_columns(df):
    df['Source IP'] = df['Source IP'].fillna('Unknown')
    df['Destination IP'] = df['Destination IP'].fillna('Unknown')
    return df

# Map 'Protocol' values using predefined map
def map_protocols(df):
    df['Protocol'] = df['Protocol'].map(PROTOCOL_MAP).fillna('Unknown')
    return df

# Apply port-based protocol mapping
def map_ports_to_protocol(df):
    def apply_port_mapping(row):
        if row['Protocol'] in ['TCP', 'UDP']:
            src_port = row['TCP Source Port'] if row['Protocol'] == 'TCP' else row['UDP Source Port']
            dest_port = row['TCP Destination Port'] if row['Protocol'] == 'TCP' else row['UDP Destination Port']
            if src_port in PORT_PROTOCOL_MAP:
                row['Protocol'] = PORT_PROTOCOL_MAP[src_port]
            elif dest_port in PORT_PROTOCOL_MAP:
                row['Protocol'] = PORT_PROTOCOL_MAP[dest_port]
        return row
    return df.apply(apply_port_mapping, axis=1)

# Clean the dataframe and prepare for analysis
def preprocess_data(df):
    df['Timestamp'] = df['Timestamp'].apply(clean_and_parse_timestamp)
    df = add_time_features(df)
    df = clean_total_length(df)
    df = clean_ip_columns(df)
    df = map_protocols(df)
    df = map_ports_to_protocol(df)
    return df
```

```python
# Descriptive statistics for 'Total Length'
def total_length_stats(df):
    stats = df['Total Length'].describe()
    print("\nDescriptive Statistics for Total Length:")
    print(stats)
    return stats

# Outlier detection using IQR
def detect_outliers(df):
    Q1 = df['Total Length'].quantile(0.25)
    Q3 = df['Total Length'].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = df[(df['Total Length'] < lower_bound) | (df['Total Length'] > upper_bound)]
    print(f"\nNumber of outliers: {len(outliers)}")
    print(outliers.head(10))
    return outliers, lower_bound, upper_bound

# Plot distributions
def plot_total_length_distribution(df):
    plt.figure(figsize=(10, 6))
    plt.subplot(1, 2, 1)
    sns.histplot(df['Total Length'], bins=50, kde=True, color='blue')
    plt.title('Total Length Distribution')

    plt.subplot(1, 2, 2)
    sns.boxplot(x=df['Total Length'], color='orange')
    plt.title('Total Length Boxplot')
    plt.xlabel('Total Length')
    plt.ylabel('Frequency')
    plt.tight_layout()
    plt.show()

    # Output of the Total Length Distribution and Boxplot
    print("\nVisual Output for Total Length Distribution and Boxplot:")
    print(f"- The histogram shows the overall distribution of total length values.")
    print(f"- The boxplot highlights the spread and identifies potential outliers in the data.")

# Hourly Total Length Statistics
def plot_hourly_total_length(df):
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x='Hour', y='Total Length', data=df, color='green')
    plt.title('Total Length vs. Hour of the Day')
    plt.xlabel('Hour of the Day')
    plt.ylabel('Total Length')
    plt.show()

    # Output of Total Length vs. Hour of Day
    print("\nVisual Output for Total Length vs. Hour of the Day:")
    hourly_stats = df.groupby('Hour')['Total Length'].mean()
    print(f"- The scatter plot shows the relationship between total length and the hour of the day.")
    print(f"- Here are the average total lengths per hour:\n{hourly_stats}")

# Frequency of each protocol
def plot_protocol_frequency(df):
    protocol_counts = df['Protocol'].value_counts()
    plt.figure(figsize=(10, 6))
    sns.barplot(x=protocol_counts.index, y=protocol_counts.values, palette='viridis', hue=protocol_counts.index)
    plt.title('Frequency Distribution of Protocols')
    plt.xticks(rotation=90)
    plt.xlabel('Protocol')
    plt.ylabel('Frequency')
    plt.show()

    # Output of Protocol Frequency
    print("\nVisual Output for Protocol Frequency Distribution:")
    print(f"- The bar chart shows the frequency distribution of protocols.")
    print(f"- Here are the counts for each protocol:\n{protocol_counts}")
```

```python
# Traffic by weekday
def plot_traffic_by_weekday(df):
    weekday_traffic = df.groupby('DayOfWeek')['Total Length'].sum().reset_index()
    plt.figure(figsize=(10, 6))
    sns.barplot(x=weekday_traffic['DayOfWeek'], y=weekday_traffic['Total Length'],
                palette='viridis', hue=weekday_traffic['DayOfWeek'])
    plt.title('Traffic Volume by Day of Week')
    plt.xlabel('Day of Week')
    plt.ylabel('Total Length')
    plt.xticks(ticks=range(7), labels=["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"])
    plt.show()

    # Output of Traffic by Weekday
    print("\nVisual Output for Traffic Volume by Day of Week:")
    print(f"- The bar chart shows the total traffic volume for each day of the week.")
    print(f"- Traffic patterns per weekday:\n{weekday_traffic}")

# Traffic volume by hour
def plot_traffic_by_hour(df):
    hourly_traffic = df.groupby('Hour')['Total Length'].sum().reset_index()
    plt.figure(figsize=(10, 6))
    sns.lineplot(x=hourly_traffic['Hour'], y=hourly_traffic['Total Length'], color='purple')
    plt.title('Traffic Volume by Hour of the Day')
    plt.xlabel('Hour of the Day')
    plt.ylabel('Total Length')
    plt.show()

    # Output of Traffic Volume by Hour
    print("\nVisual Output for Traffic Volume by Hour of the Day:")
    print(f"- The line plot shows the total traffic volume aggregated by each hour.")
    print(f"- Hourly traffic summary:\n{hourly_traffic}")

# Main function to run all steps
def main(file_path):
    df = load_data(file_path)
    if df is not None:
        df = preprocess_data(df)

        # Descriptive statistics and outlier detection
        total_length_stats(df)
        outliers, lower_bound, upper_bound = detect_outliers(df)

        # Plotting and visual outputs
        plot_total_length_distribution(df)
        plot_hourly_total_length(df)
        plot_protocol_frequency(df)
        plot_traffic_by_weekday(df)
        plot_traffic_by_hour(df)

# Run the analysis with the given file path
file_path = 'Documents/pcap_files/all_extracted_data.csv'  # Update with your actual file path
main(file_path)
```

Statistical Metrics

Mean Packet Length (bytes)
The mean packet length is the average size of the packets being transmitted, and it reflects the overall traffic type.

Min Length (bytes)
The minimum packet length helps us understand the smallest packet transmitted during the botnet activity. Small values (close to 0) might indicate ping-like activity (ICMP packets, for example).

Max Length (bytes)
The maximum packet length is useful for identifying anomalies or large-scale operations like data exfiltration or payload delivery.

Standard Deviation (bytes)
Standard deviation measures the variability or spread of packet sizes. A higher standard deviation suggests that the botnet generates a wide range of packet sizes, possibly due to multiple types of attack methods being employed.

Botnet	Mean Packet Length (bytes)	Min Length (bytes)	Max Length (bytes)	Standard Deviation (bytes)
Neris	203	29	8,905	542
Rbot	823	39	2,960	726
Virut	642	40	9,453	808
DonBot	182	40	2,953	456
Sogou	880	40	8,728	1,428
Murlo	217	32	7,340	411
NSIS.ay	416	36	19,021	936

Protocol Distribution Findings

- **TCP:** Dominant across Neris, Virut, and Sogou for stable communication
- **UDP:** Heavily used by Rbot for high-volume DDoS attacks
- **SMTP:** Found in DonBot, used for spam-based operations (59.9% of traffic)
- **ICMP:** accounts for (48.5%) of Rbot traffic, reflecting its role in diagnostic and network reconnaissance activities
- **SNMP:** usage is prominent in Rbot (45.7%), suggesting possible exploitation of network monitoring systems
- **HTTP:** dominates in NSIS.ay, Sogou, and Virut (46.7%, 32.1%, 23.8%) for C&C communication
- **HTTPS:** plays a major role in DonBot, Neris, and Virut (22.7%, 22.3%, 18.6%), likely for encrypted data transmission

Protocol Distribution Across Botnets:

Protocol	Neris (%)	Rbot (%)	Virut (%)	DonBot (%)	Sogou (%)	Murlo (%)	NSIS.ay (%)	Combined (%)
TCP	53.5%	0.2%	55.1%	23.7%	53.2%	44.3%	30.2%	38.5%
UDP	0.12%	53.7%	0.18%	0.23%	0.5%	14.1%	7.1%	14.2%
HTTP	13.2%	0.02%	23.8%	0.2%	32.2%	5.6%	46.7%	25.4%
DNS	4.4%	0.01%	0.2%	0.3%	9.3%	11.1%	0.4%	3.7%
HTTPS	22.3%	0.01%	18.6%	22.7%	6.2%	5.3%	5.4%	11.4%
SMTP	17.8%	0.008%	0.23%	59.9%	0.1%	4.1%	4.9%	9.7%
IRC	3.2%	0.12%	1.3%	0.3%	6.1%	3.3%	1.3%	1.7%
ICMP	0.43%	48.5%	0.01%	0.15%	0.1%	0.08%	0.02%	22.6%
SNMP	0.37%	45.7%	0.01%	0.12%	0.07%	0.14%	0.11%	7.4%
Unknown	0.37%	0.08%	0.16%	1.2%	0.33%	1.06%	0.11%	0.92%

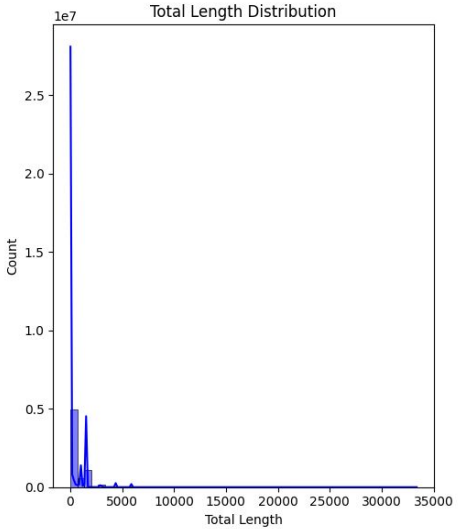
Outlier Patterns

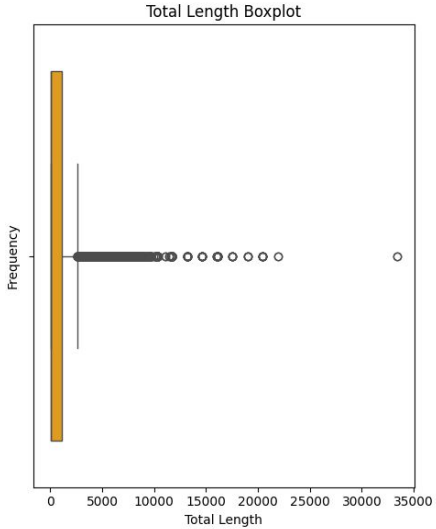
Malicious traffic shows high variability and outliers in packet sizes, unlike safe traffic:

Outliers in Packet Length:

Neris (botnet-capture-20110810-neris.pcap): *14495 outliers*
Neris (botnet-capture-20110811-neris.pcap): *14152 outliers*
Rbot (botnet-capture-20110812-rbot.pcap): *0 outliers*
Rbot (botnet-capture-20110815-rbot-dos.pcap): *0 outliers*
Virut (botnet-capture-20110815-fast-flux.pcap): *374 outliers*
Virut (botnet-capture-20110815-fast-flux-2.pcap): *19600 outliers*
Donbot (botnet-capture-20110816-donbot.pcap): *8779 outliers*
Sogou (botnet-capture-20110816-sogou.pcap): *1125 outliers*
Murlo (botnet-capture-20110816-qvod.pcap): *10929 outliers*
Neris (botnet-capture-20110817-bot.pcap): *22561 outliers*
Rbot (botnet-capture-20110818-bot-2.pcap): *1203 outliers*
NSIS.ay (botnet-capture-20110819-bot.pcap): *5075 outliers*

Graphs from Combined Data:





Temporal Traffic Patterns

2.3.2 Temporal Analysis

Temporal Analysis examines the time-based patterns in botnet traffic, focusing on when malicious activities peak or fluctuate. Hourly and weekly traffic volumes identify specific times of day or days of the week when botnets are most active, such as DDoS attacks during business hours or data exfiltration during off-peak times:

Hourly Traffic

Botnet	Peak Hour	Mean Packet Length (bytes)	Total Volume (bytes)
Neris	21:00–22:00	281	20,124,723
Rbot	20:00–21:00	823	18,700,600
Virut	02:00–03:00	649	29,443,213
DonBot	20:00–21:00	268	4,515,687
Sogou	23:00–00:00	881	18,169,111
Murlo	12:00–13:00	392	18,618,405
NSIS.ay	00:00–01:00	526	40,595,958

Hourly Traffic

- *Botnets show distinct peak hours, reflecting a mix of automated and human-triggered operations.*
- *Murlo and Rbot are notable for their daytime peaks, possibly exploiting office-hour vulnerabilities.*

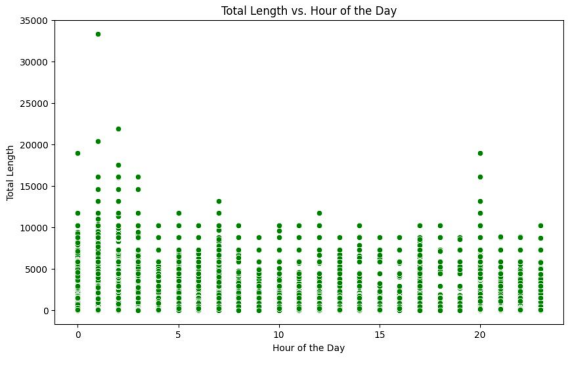
Weekly Traffic

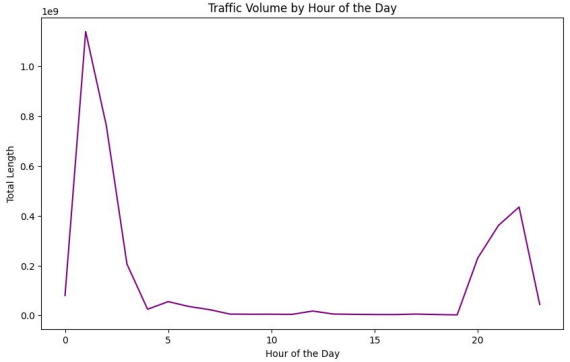
Botnet	Day of Maximum Activity	Total Volume (bytes)
Neris	Thursday	20,124,723
Rbot	Friday	18,700,600
Virut	Wednesday	29,443,213
DonBot	Thursday	4,515,687
Sogou	Thursday	18,169,111
Murlo	Friday	18,618,405
NSIS.ay	Friday	40,595,958

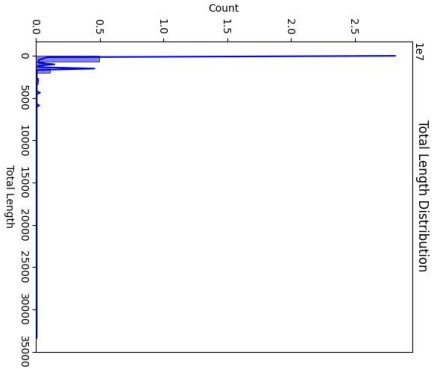
Weekly Traffic

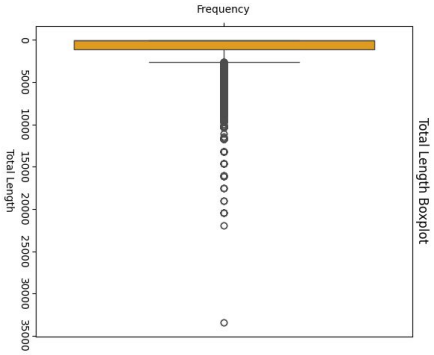
- *Peaks on Thursday and Friday suggest malicious actors prioritize operations before weekends.*
- *Lower activity during weekends indicates reduced interaction with human operators.*

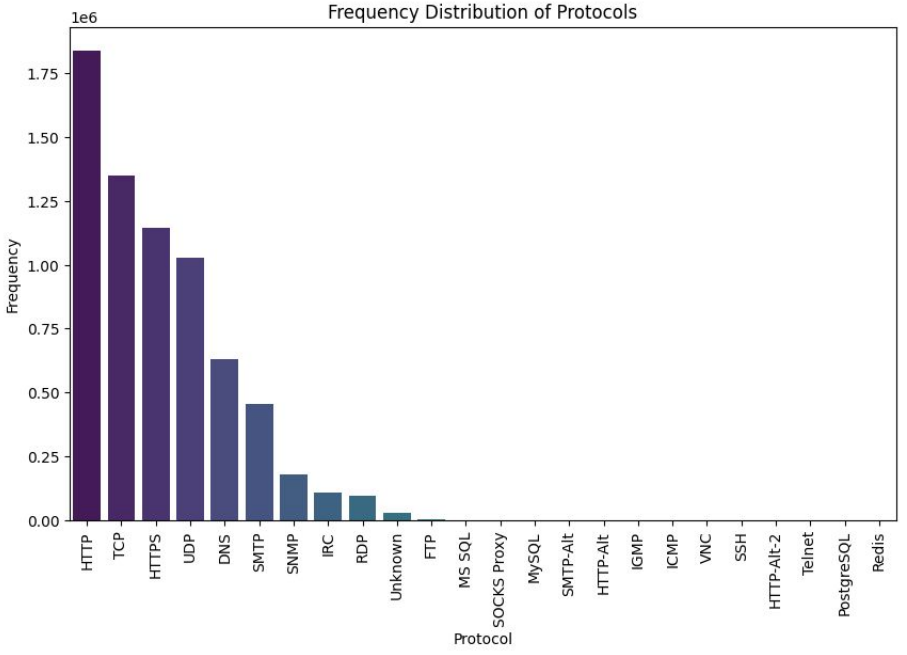
Visualizing Collective Differences Analysing All Scenarios





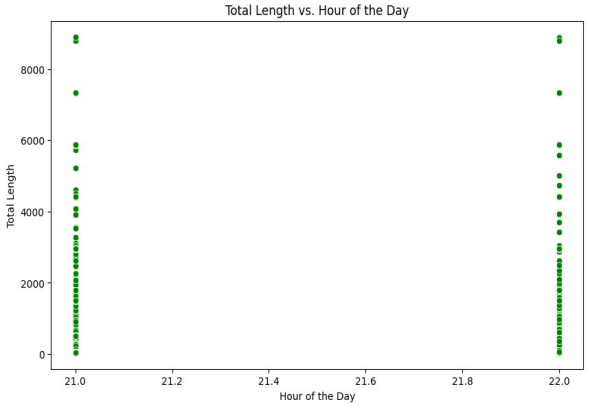




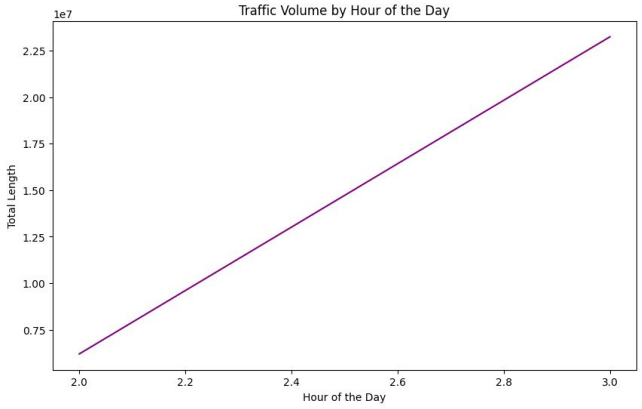


Visualizing Differences (Safe vs Malicious Traffic)

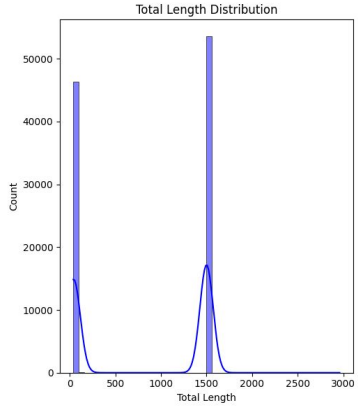
Dataset Neris (botnet-capture-20110810-neris.pcap):

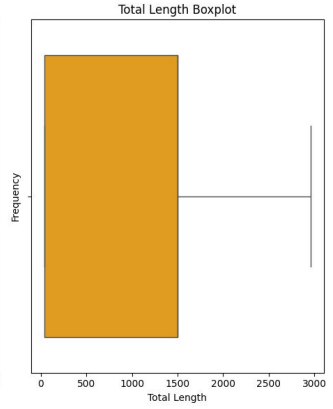


Dataset Virut (botnet-capture-20110815-fast-flux.pcap):

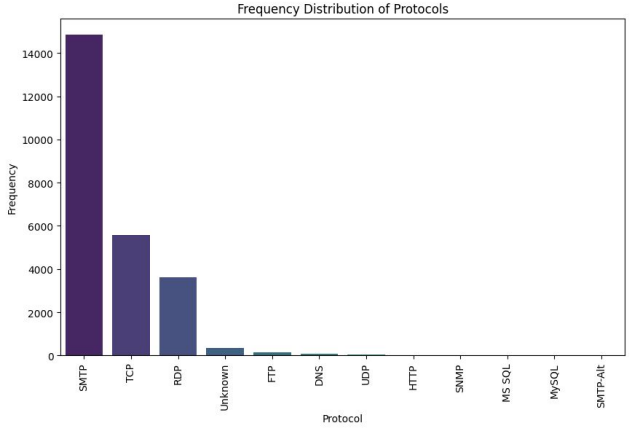


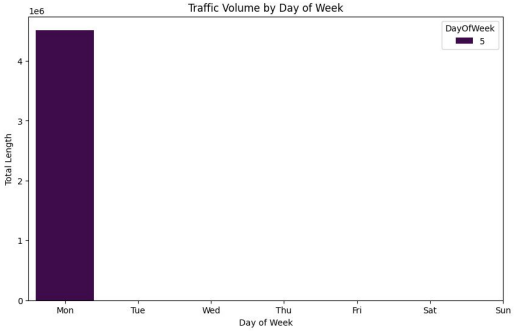
Dataset Rbot (botnet-capture-20110815-rbot-dos.pcap):

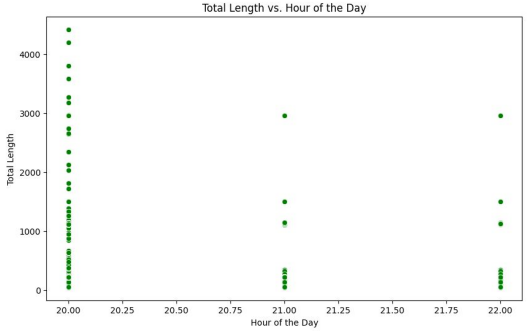


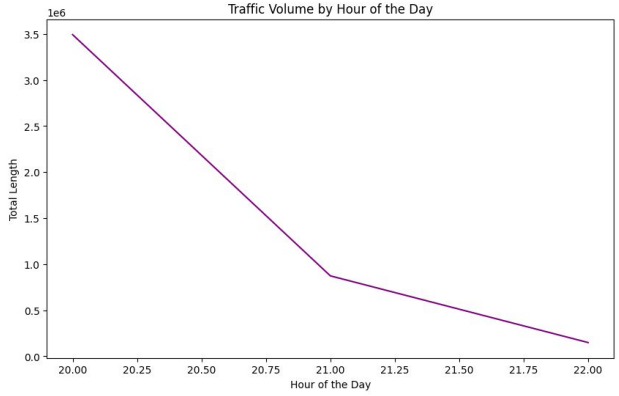


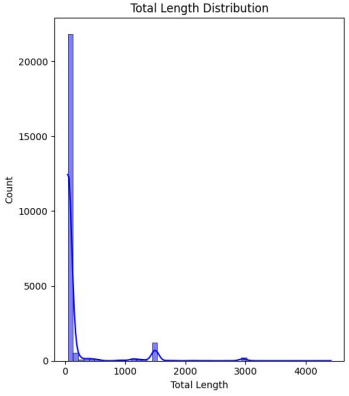
Dataset DonBot (botnet-capture-20110816-donbot.pcap):

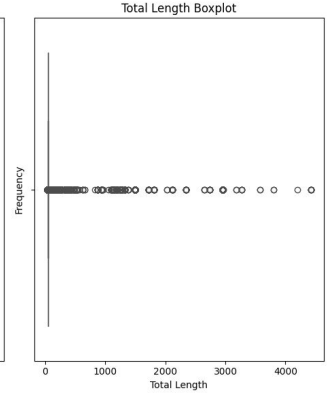




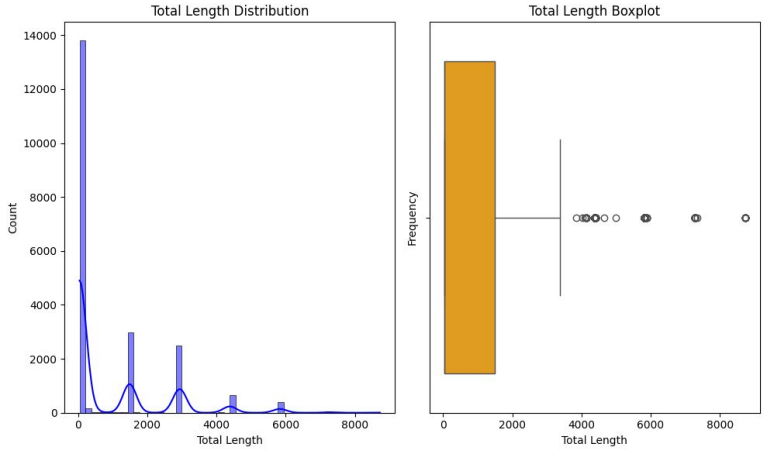


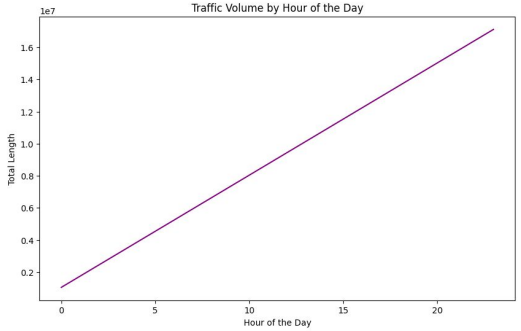


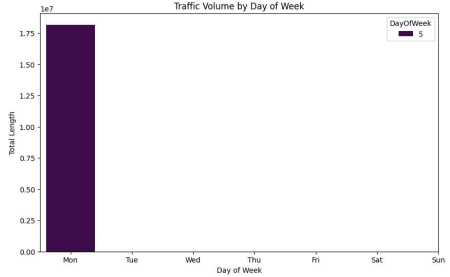


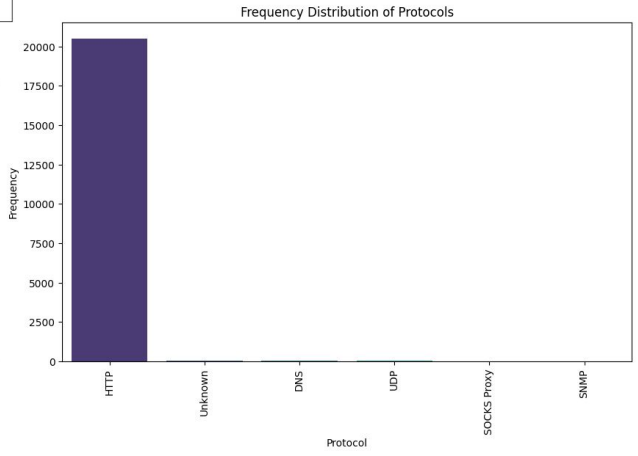


Dataset Sogou (botnet-capture-20110816-sogou.pcap):

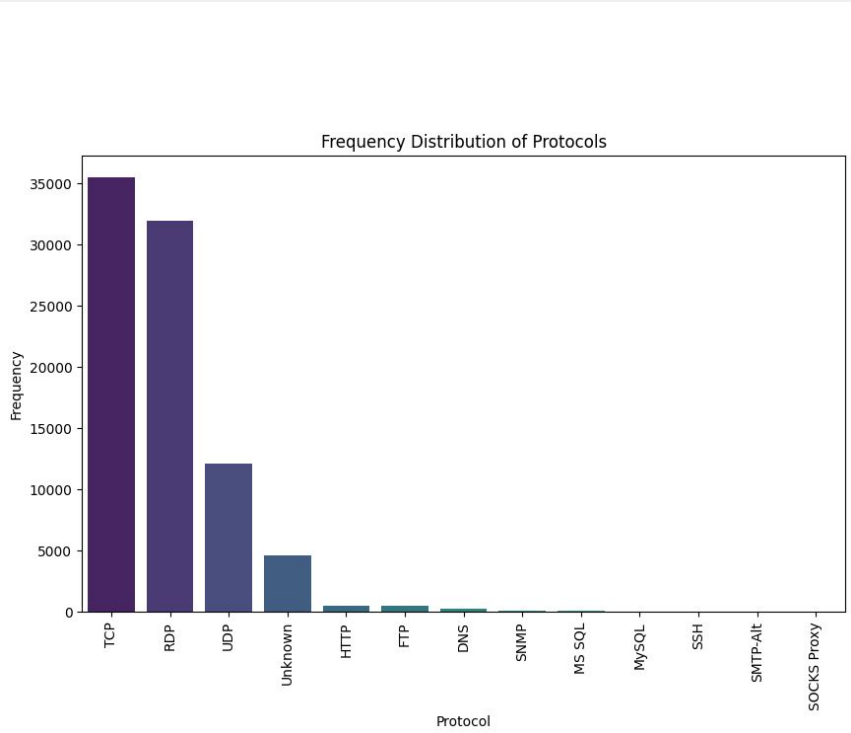


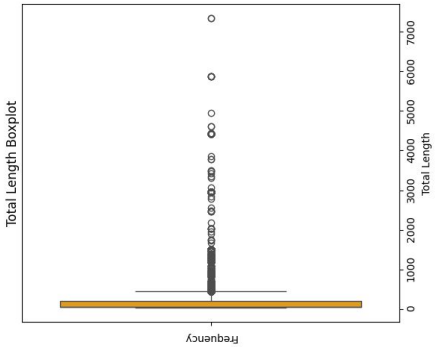


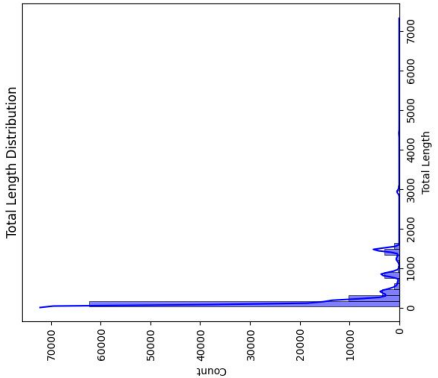


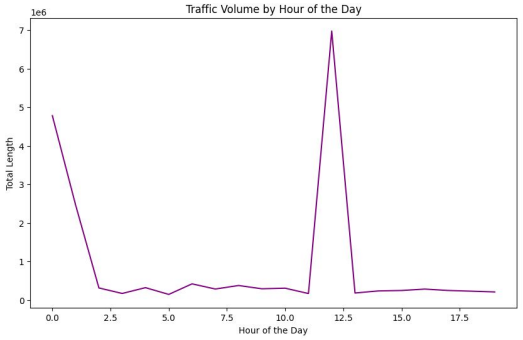


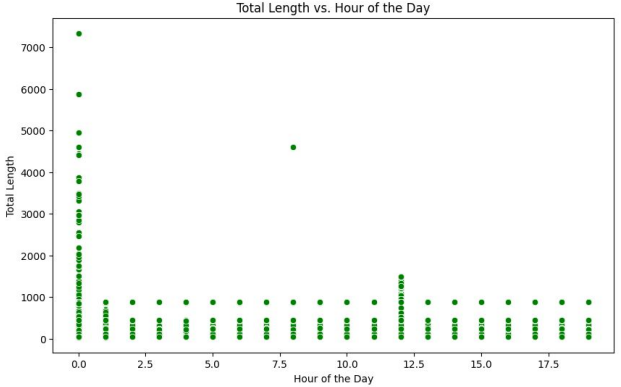
Dataset Murlo (botnet-capture-20110816-qvod.pcap):



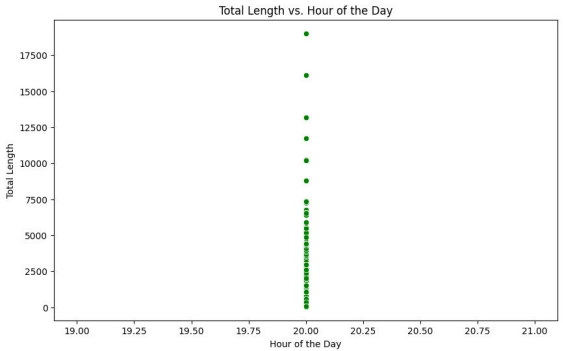


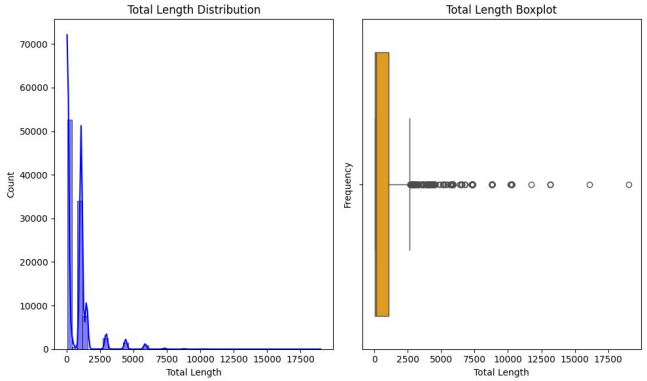


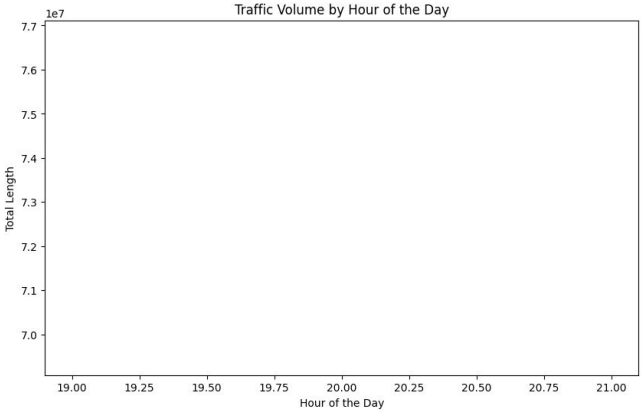




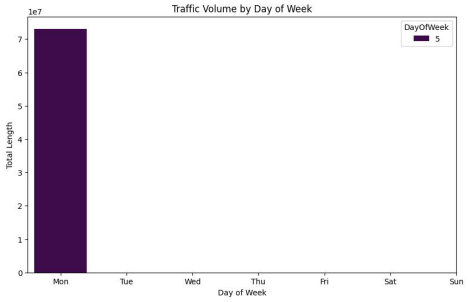
Dataset NSIS.ay (botnet-capture-20110819-bot.pcap):

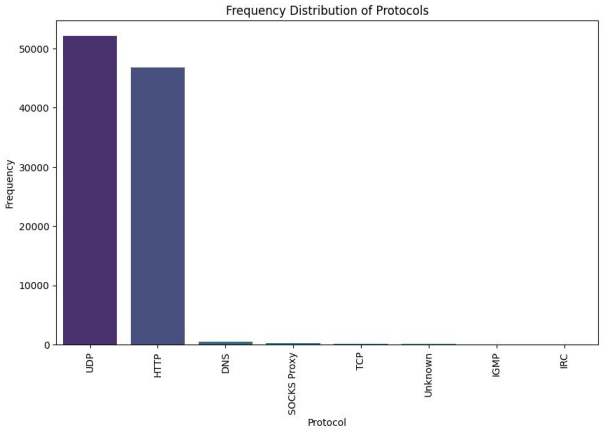






Hour	Total Length
20	73097271.0





Conclusions

- The CTU-13 dataset reveals clear distinctions between botnet and normal traffic through protocol usage, packet size variability, and temporal patterns
- Insights into specific botnet behaviors provide a foundation for improving detection techniques
- Understanding botnet characteristics enhances network resilience and reduces the risk of large-scale attacks

Recommendations

Detection Strategies

Implement anomaly-based monitoring to flag packet size outliers and protocol misuse

Focus on high ICMP/UDP traffic for detecting DDoS botnets like Rbot

Monitor DNS and HTTP traffic for signs of C&C communication

Future Enhancements

Develop machine learning models to predict botnet activity based on traffic features

Incorporate Pearson's Correlation Coefficient