Universidade de Aveiro
Mestrado em Cibersegurança
Código: 41782 - Segurança em Redes de Comunicações
Responsible: Paulo Jorge Salvador Serra Ferreira

Report 2

Monday 20th May, 2024
Ricardo Covelo (102668) - Telmo Sauce (104428)

Contents

1 Introduction

In the modern digital landscape, ensuring the security of communication networks is of paramount importance. This project, guided by Professors Paulo Salvador and António Nogueira, focuses on enhancing network security through the implementation of Security Information and Event Management (SIEM) rules. By analyzing network traffic data, the goal is to detect anomalous behaviors that may indicate compromised devices or illicit activities. Through defining, testing, and refining these SIEM rules, the project aims to establish a robust cybersecurity framework capable of identifying potential threats within a corporate network.

2 Data Analisys

To begin the analysis confirmed that all TCP communications take place on port 443, which is connected to HTTPS by default, and all UDP communications take place on port 53, which is connected to DNS.

We also saw an increase in the uploaded bytes as compared to the Anomalous file. This implies that there may be data exfiltration occurring.

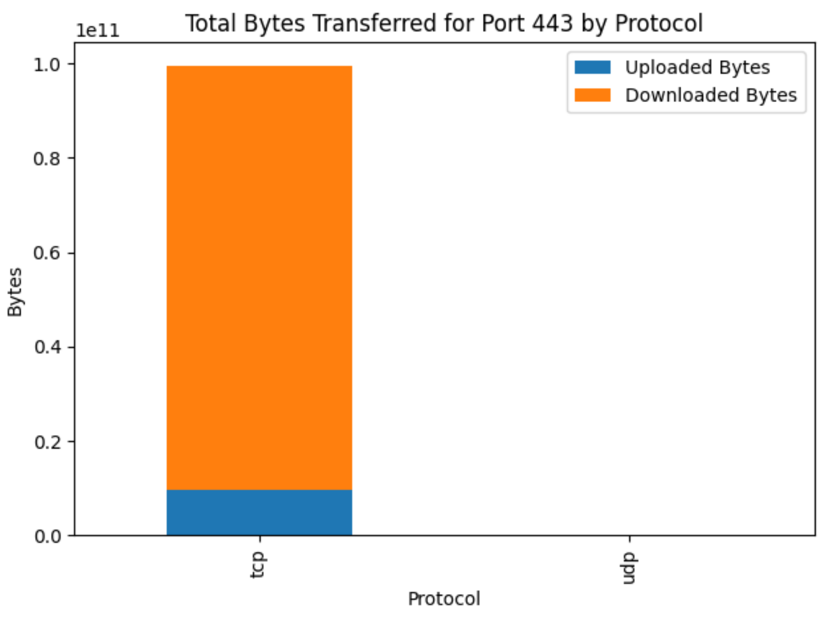


Figure 1: Bytes per protocol Normal

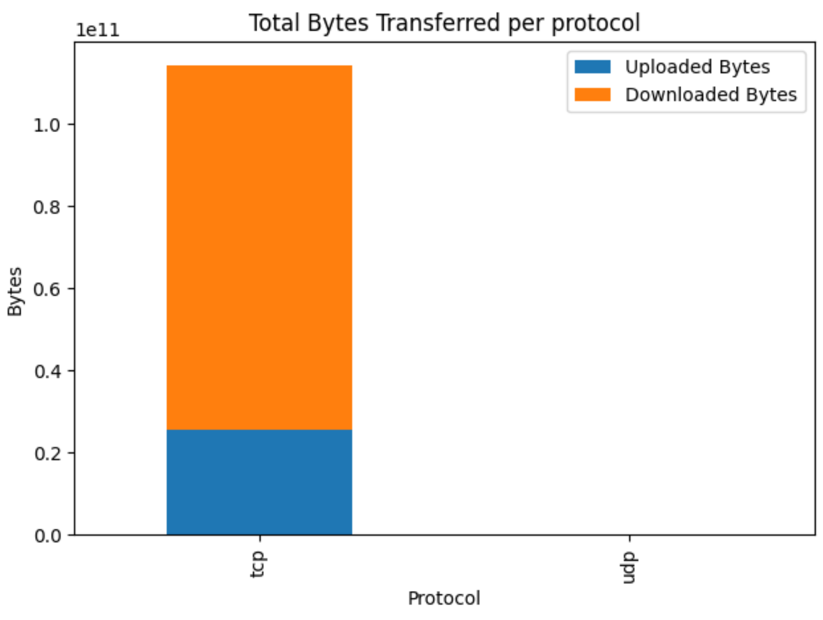


Figure 2: Bytes per protocol Anomalous

To investigate if there are any data exfiltration we can compare the 10 IPs with the most uploaded bytes to check if there are any spikes comparing the two.

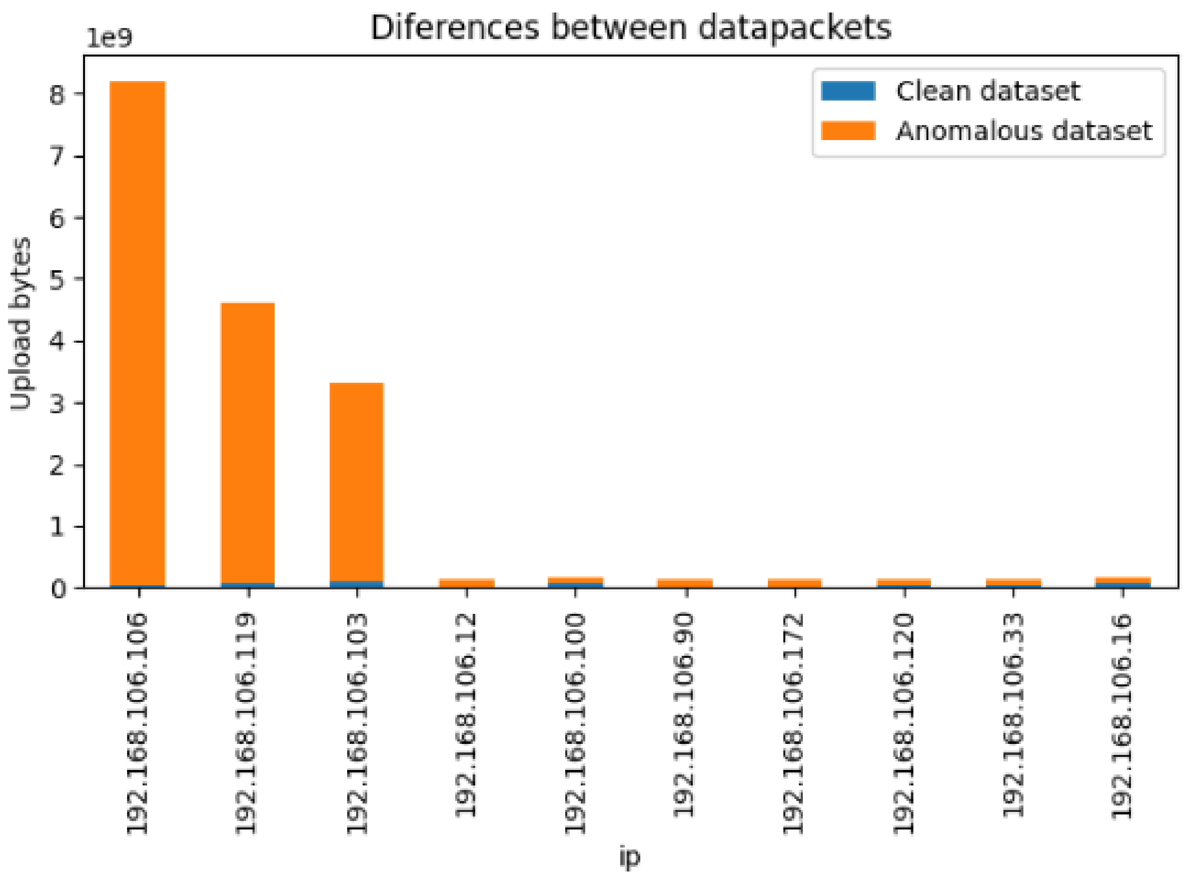


Figure 3: 10 ip with most Uploads

We can see that there was a huge difference between the results, namely the IPs 192.168.106.103, 192.168.106.106, and 192.168.106.119 might be compromised. We can check that this behavior is not normally compared with the difference in downloaded bytes.

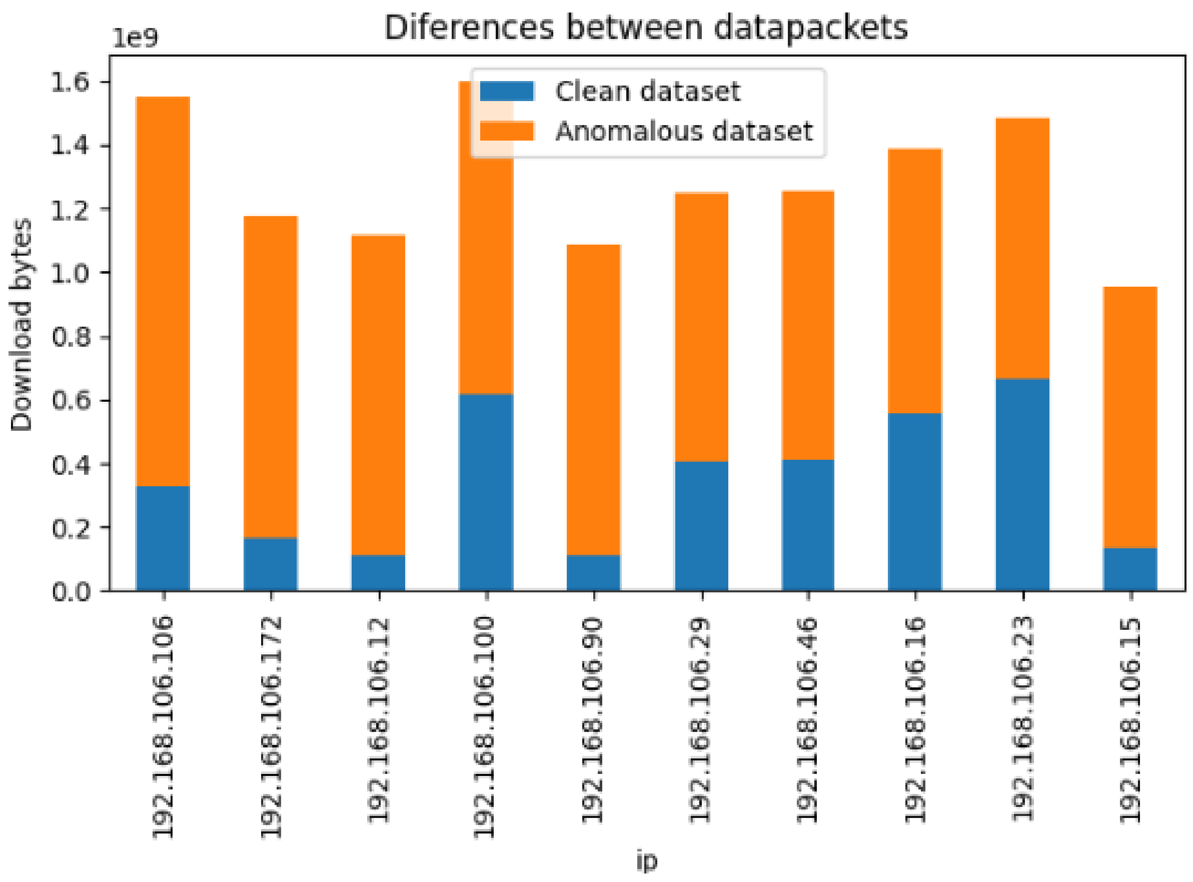


Figure 4: 10 ip with most Downloads

As seen there isn't even close to the huge difference seen before. We also checked the different behavior of countries between datasets.

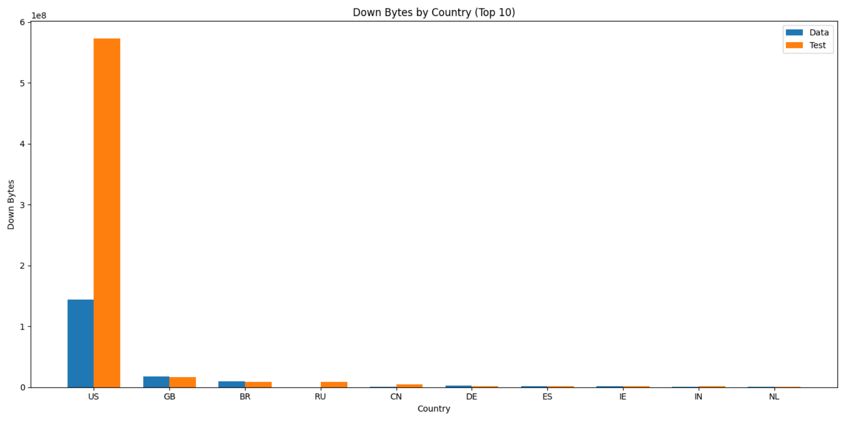


Figure 5: contries with most uploads

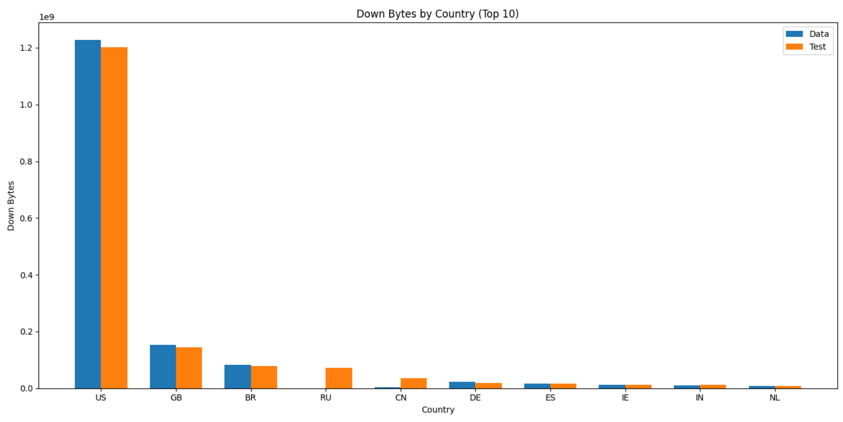


Figure 6: contries with most downloads

As seen there is a huge difference in the US-uploaded bytes, this is the possible destination of the previously mentioned data exfiltration. There are also minor changes in Russia and China that might need attention. Finally, we can compare the number of communications along the day to see if there are major discrepancies between the datasets.

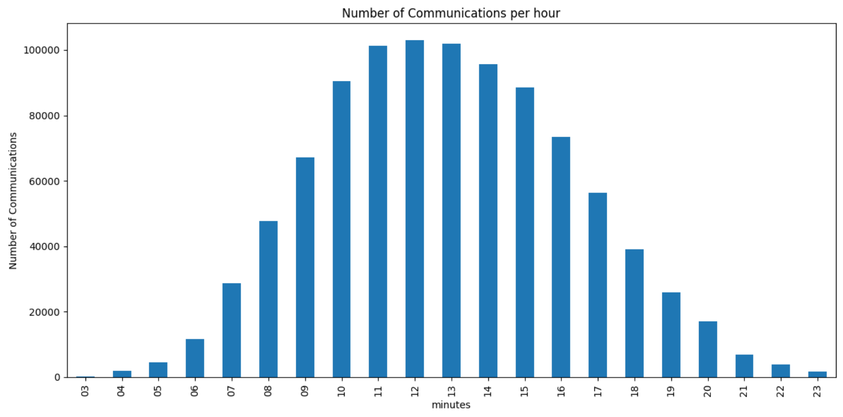


Figure 7: Normal communications per minute

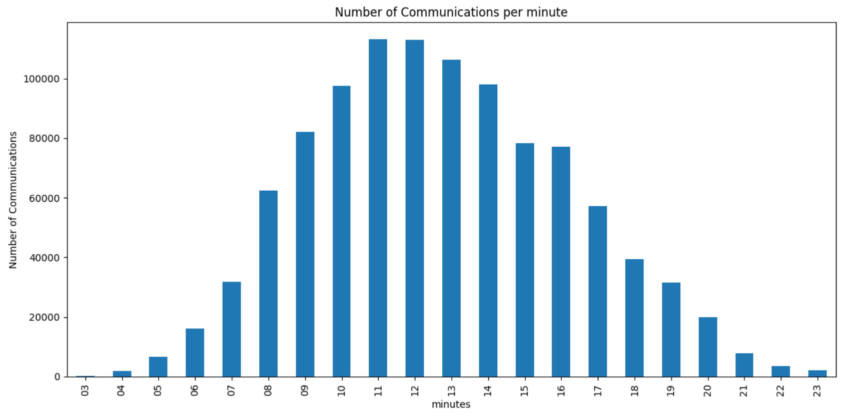


Figure 8: Anomalous communications per minute

As we can see, although the anomalous data has a few more connections, the pattern stays similar in both data packets.

2.1 Non-anomalous File

2.1.1 Internal Services

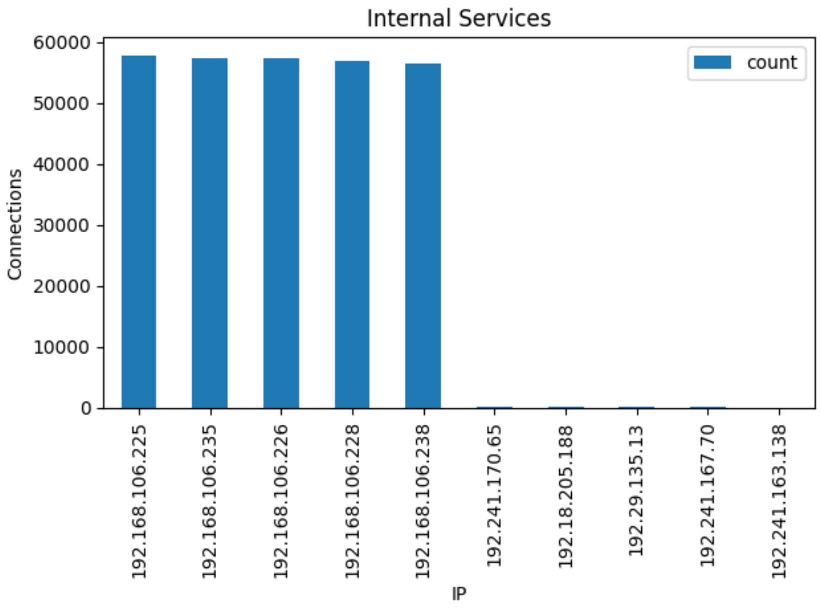


Figure 9: Internal services

The IP addresses **”192.168.106.225”, ”192.168.106.235”, ”192.168.106.226”, ”192.168.106.228”, ”192.168.106.238”** are internal IPs that have the most connections. This suggests that these IPs likely correspond to internal services within the network.

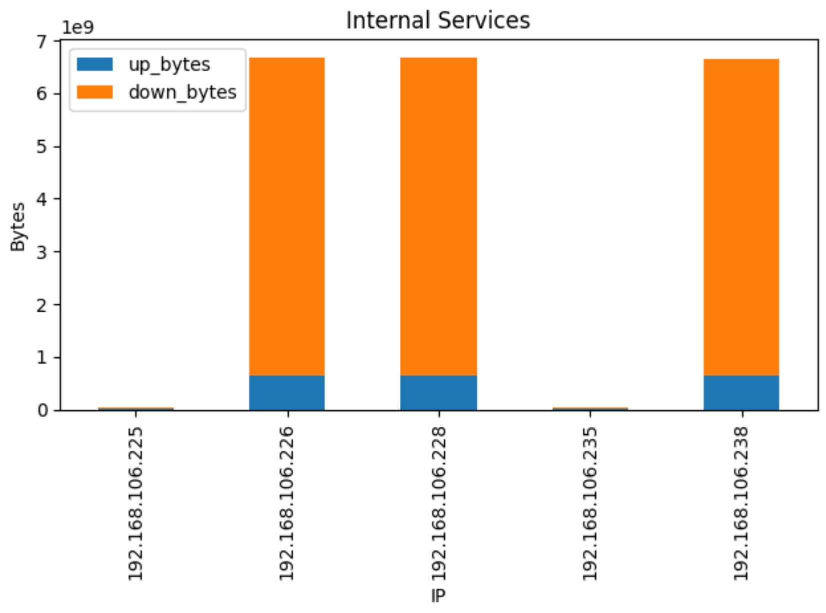


Figure 10: Internal services bytes downloaded and uploaded

Out of all the services, only three show a significant exchange of data. The type of these services may be the cause of this. For example, the download/upload bytes for a file system are usually much larger than those for a webpage. The data throughout the day also stays between normal values as we can see below, although the IPs **192.168.106.225 and 198.168.106.235** have twice the usage compared with the other internal services

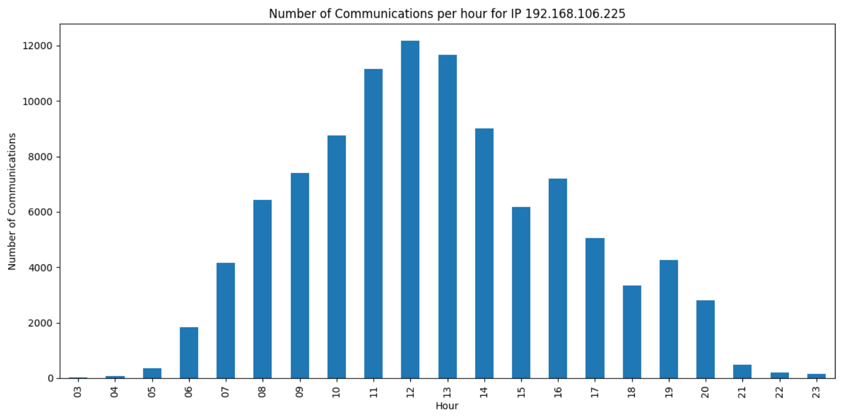


Figure 11: 192.168.106.225

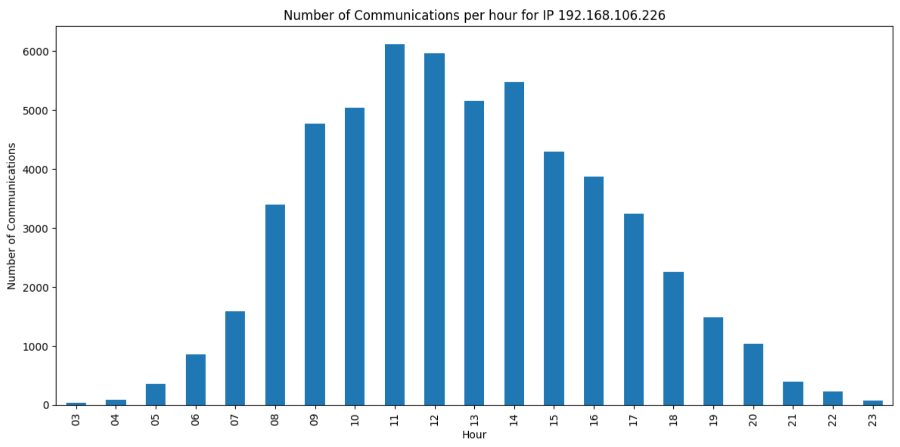


Figure 12: 192.168.106.226

Figure 13: 192.168.106.228

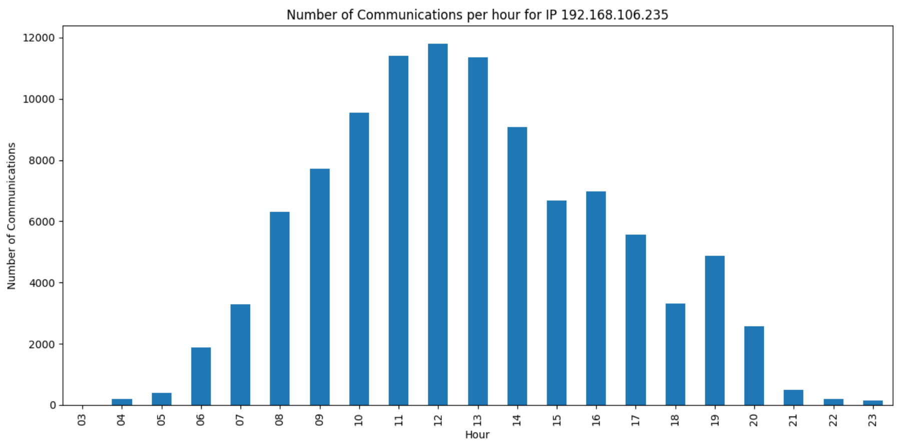


Figure 14: 192.168.106.235

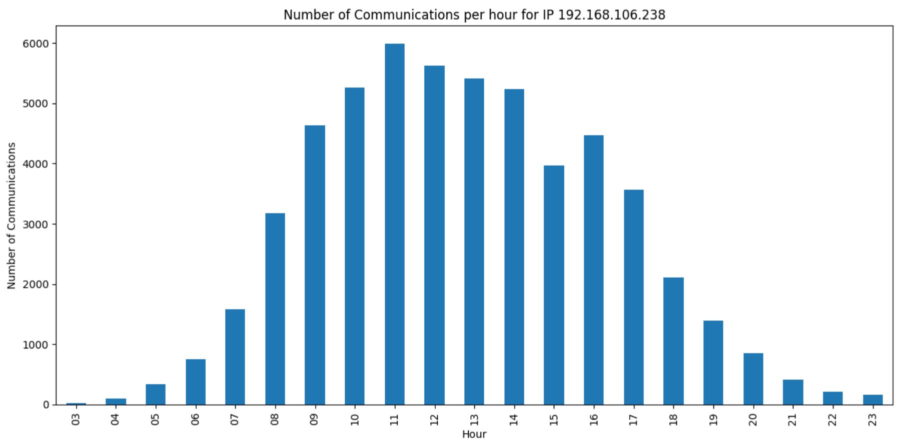


Figure 15: 192.168.106.238

2.1.2 External Services

We used similar methods to identify the external services as we did the internal services. However, we were unable to locate a distinct separation between the various IP addresses, one reason for that might be that most of the connections done to the exterior are mostly services, and only a few aren't. As there are so few non-services their data don't show up in the graph.

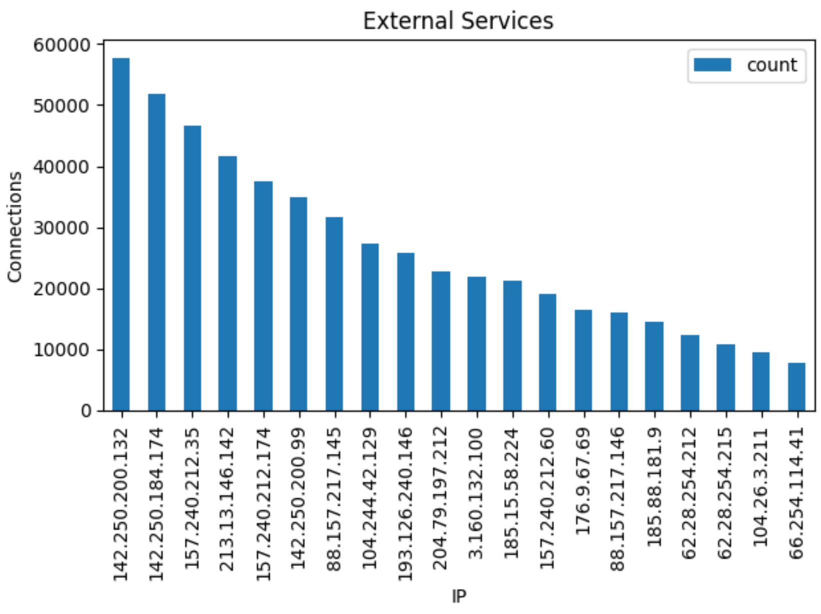


Figure 16: External services

Then, we attempted to differentiate them by the number of bytes downloaded and uploaded, but that approach had no results.

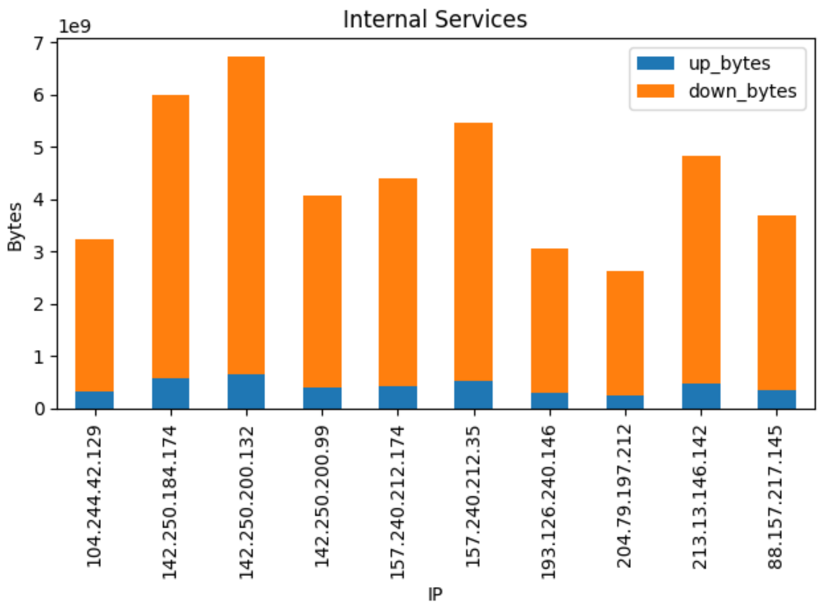


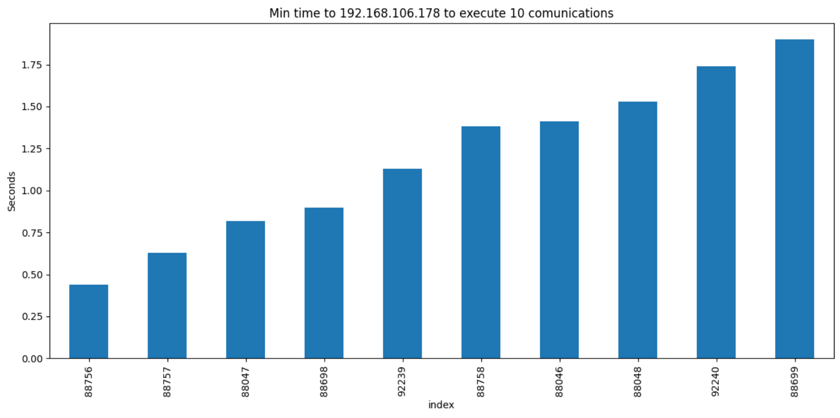
Figure 17: External services bytes downloaded and uploaded

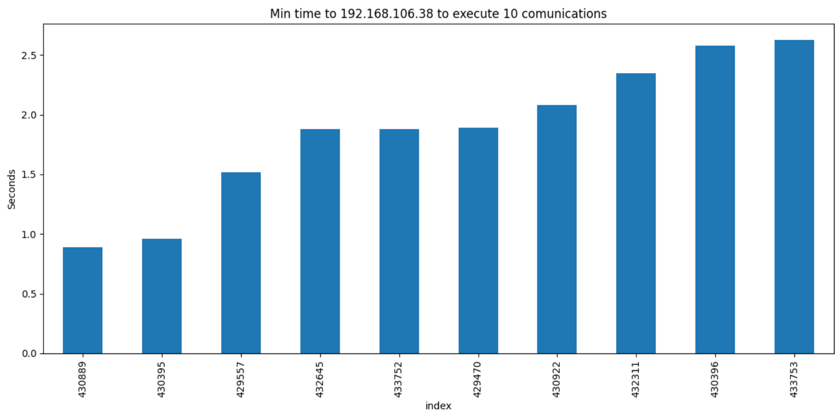
2.2 Anomalous file

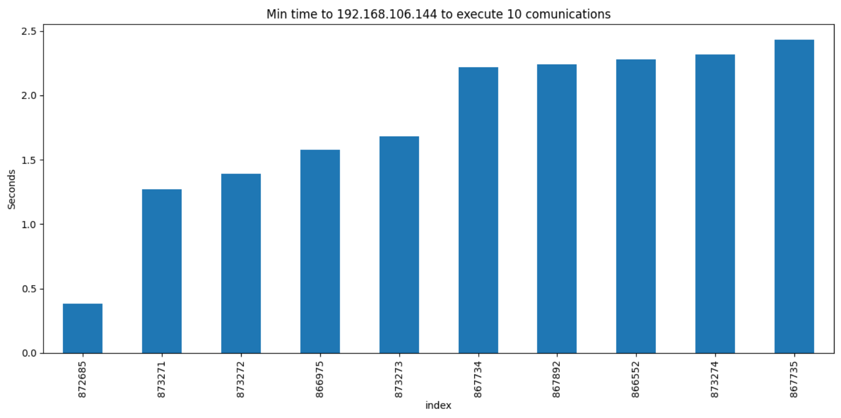
2.2.1 Botnet

Compromised devices managed by a central command are referred to as botnets. The devices that are compromised commonly called ”bots” are usually infected with malicious software without the owners’ knowledge.

We identify IP addresses that initiate new internal communications compared with the base data packet as potential suspects. **New IPs that are communicating internally: [’192.168.106.144’ ’192.168.106.178’ ’192.168.106.38’]**. As for confirming our suspicions, we saw what were the minimum time necessary to do 10 communications. The results were very fast, and very suspicious for a normal user.







2.3 Server

Analyzing the protocols, ports, and quantity of flows per IP when other entities connect to the company's exposed HTTP service server is important for detecting unusual activity and providing network security. Understanding the used protocols gives insight into the nature of communication occurring, helping the detection of possibly harmful connections. By keeping an eye on ports, one might identify any unauthorized services or strange communication patterns that might point to security risks. Monitoring the number of flows per IP helps in establishing baseline behavior and spotting changes that may indicate suspicious activity, such as a large increase in traffic compared with the normal flow coming from a particular source, which might be a sign of a Distributed Denial of Service (DDoS) attack or enumeration attempts.

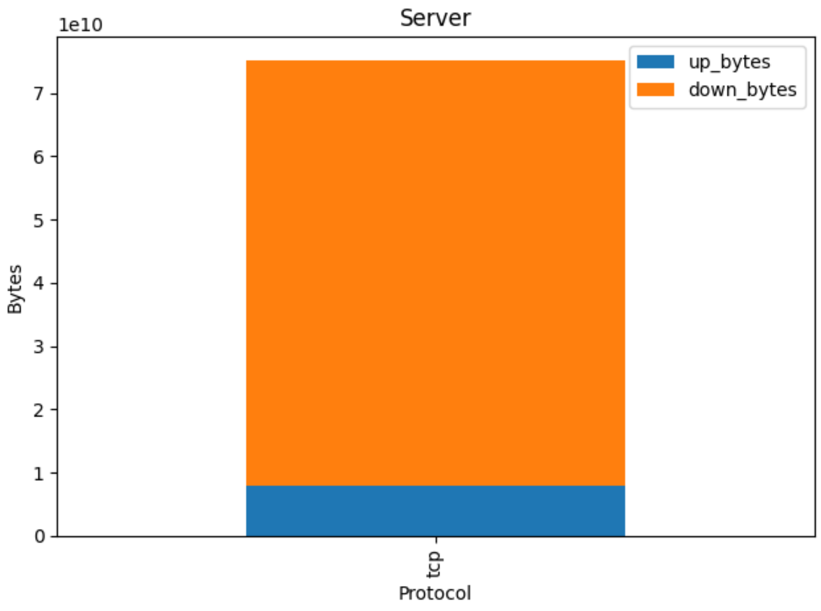


Figure 18: Bytes per protocol in Public Servers

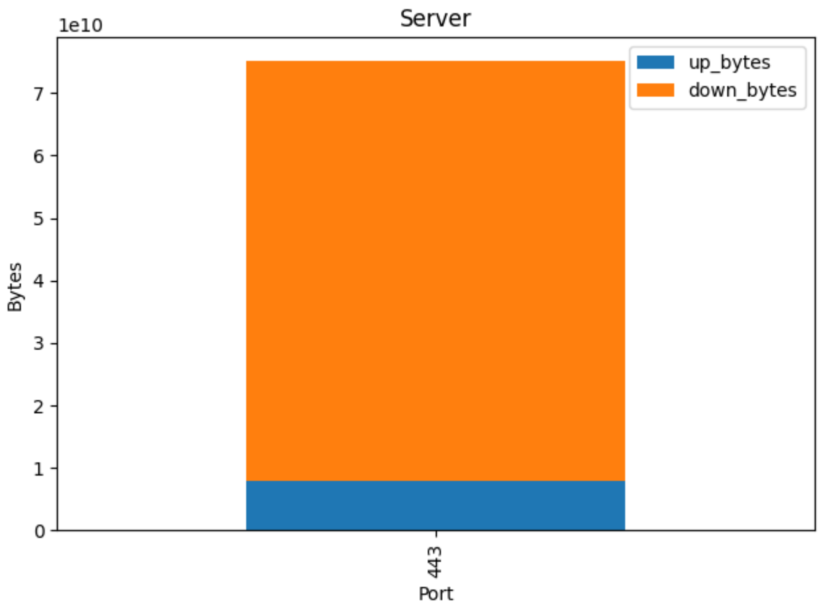


Figure 19: Bytes per Port in exposed Servers

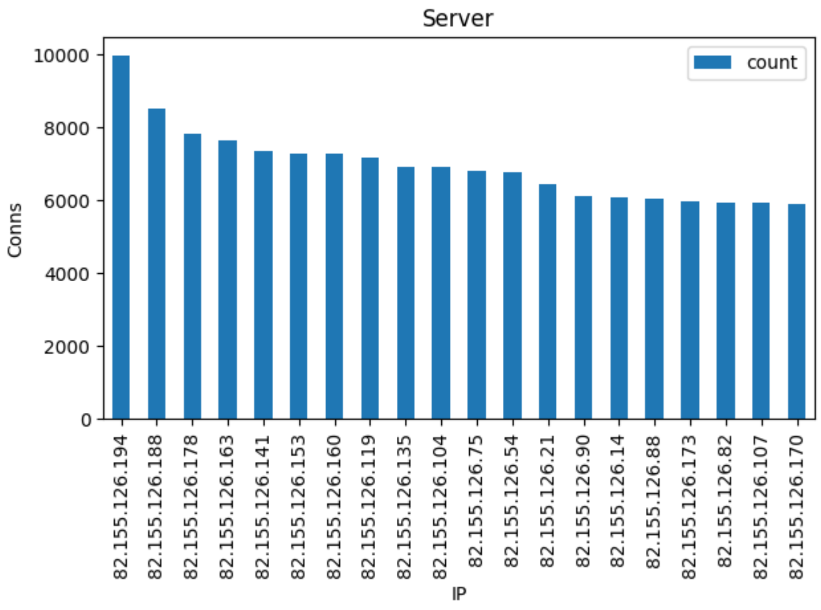


Figure 20: Connections per IP in public servers

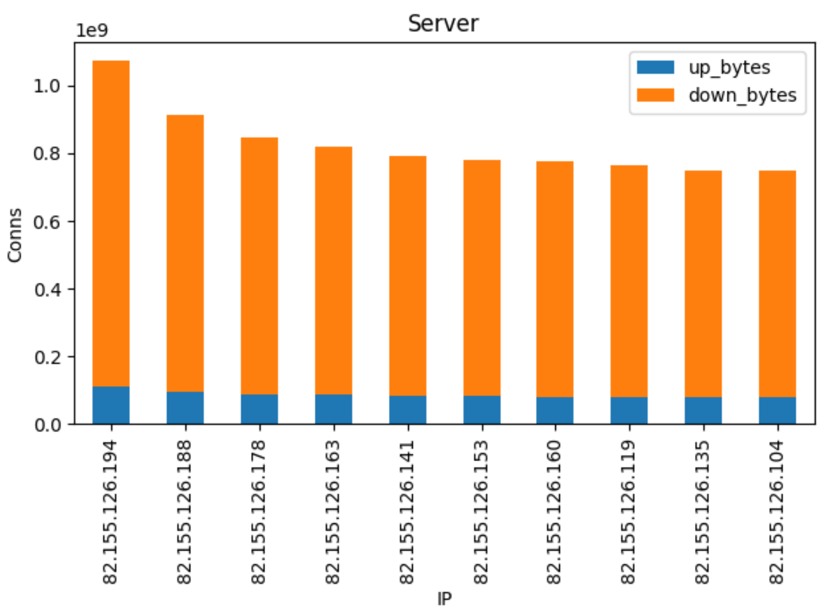


Figure 21: Bytes per IP in public Servers

As seen above no data is pointing to any of the previously mentioned activities, no major changes in traffic amount, and no use of unusual ports or protocols. There was a possibility that the connections were made by weird IPs of potentially malicious countries or organizations but all connections are made in Portugal by MEO eliminating that possibility. We thought then that it might be an attack of DDOS, where the IP trying to make a lot of communications overloading the server, but that looks like it is not the case either.

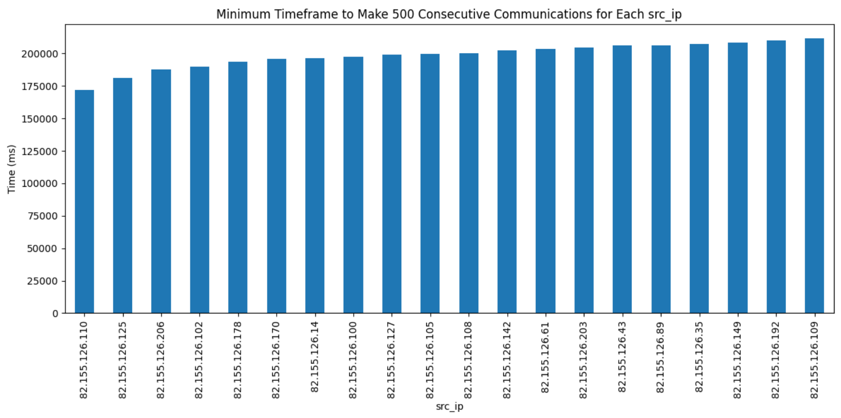


Figure 22: Time needed for 500 comunications

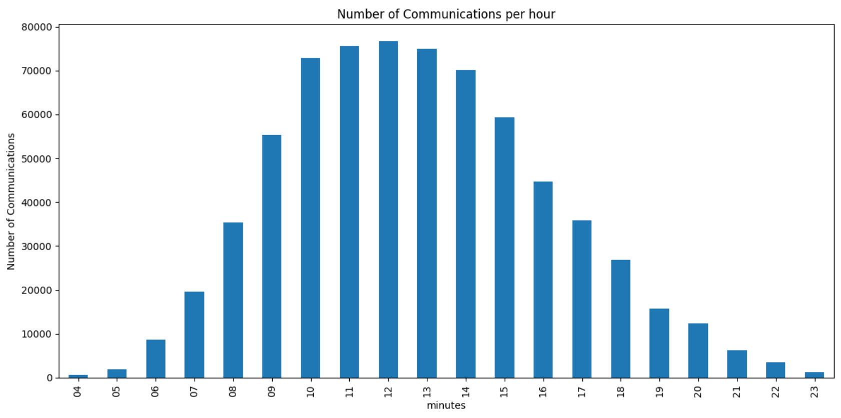


Figure 23: Comunications per Hour

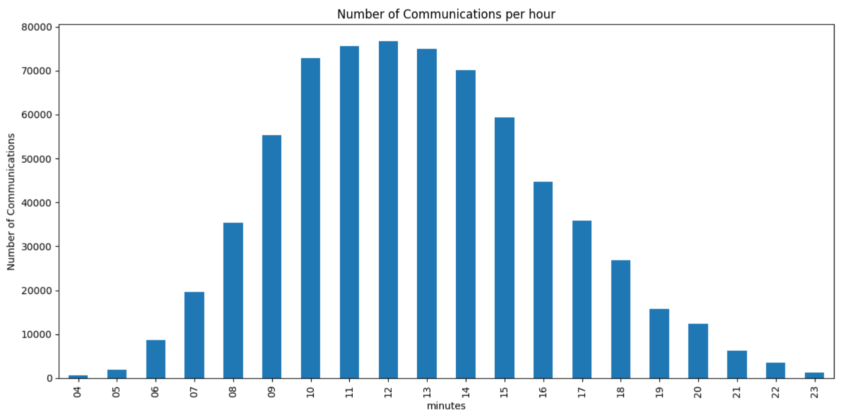


Figure 24: Sum of Uploaded and Downloaded bytes per Hour

As we can see there are no indications of a DDOS Attack since there are no major spikes in data or communication and both graphs have the expected shape(Standard Deviation). Also, the time to make 500 communications isn't low enough for any of the IPs to cause suspicion.

3 Rules

The rules developed in this project can be categorized into three main groups: those that detect country behaviors, those that detect individual IP behaviors, and those that detect network connections or requests.

- **Country-Level Behaviors**: These rules analyze traffic patterns associated with specific countries to identify potential anomalies.
- **Individual IP Behaviors**: These rules focus on the activity of individual IP addresses, looking for unusual patterns or behaviors.

- **Network Connections/Requests**: These rules examine the connections or requests between devices to detect irregularities.

Each of these rule categories can be then divided into 3 more categories for detecting the 3 possible attacks.

For Command and Control (C&C) Detection, the rules focus on the download data of the packets.

For Data Exfiltration Detection Rules aim at identifying exfiltration look at the uploaded data.

Botnet Activity Detection rules are concerned with the number of connections made by a device, especially between internal Devices.

Finally, the rules will use the information from the non-anomalous dataset to create context.

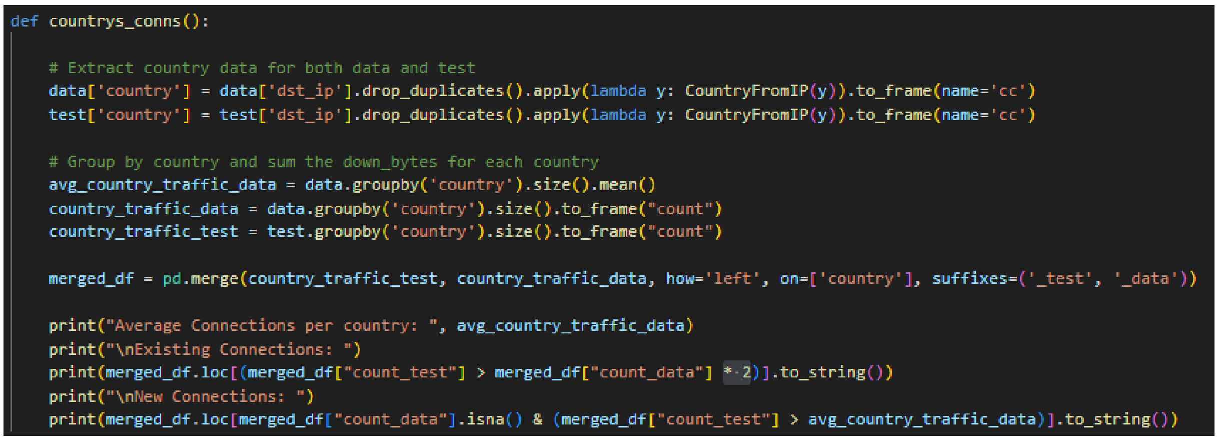
For example, in the rules aimed at detecting individual IP behaviors, the system will first check if the IP exists in the clean dataset. If the IP is found, the current values will be compared to those of the existing IP. If the IP is not present in the dataset, the values will be compared to the average. This concept is similarly applied to the other two types of rules.

3.1 country behaviors

The following rules are trying to detect trends from countries that can be attacking the organization, this helps the organization focus its attention more on IPs coming from those locations.

3.1.1 countrys_conns

The following rule aims to detect connections in a broader view where we focus on the country making each request. With this rule Countries where the number of connections went above the standard can be flagged, since it can indicate malicious activity.



```python
def countrys_conns():

    # Extract country data for both data and test
    data['country'] = data['dst_ip'].drop_duplicates().apply(lambda y: CountryFromIP(y)).to_frame(name='cc')
    test['country'] = test['dst_ip'].drop_duplicates().apply(lambda y: CountryFromIP(y)).to_frame(name='cc')

    # Group by country and sum the down_bytes for each country
    avg_country_traffic_data = data.groupby('country').size().mean()
    country_traffic_data = data.groupby('country').size().to_frame("count")
    country_traffic_test = test.groupby('country').size().to_frame("count")

    merged_df = pd.merge(country_traffic_test, country_traffic_data, how='left', on=['country'], suffixes=('_test', '_data'))

    print("Average Connections per country: ", avg_country_traffic_data)
    print("\nExisting Connections: ")
    print(merged_df.loc[(merged_df["count_test"] > merged_df["count_data"] * 2)].to_string())
    print("\nNew Connections: ")
    print(merged_df.loc[merged_df["count_data"].isna() & (merged_df["count_test"] > avg_country_traffic_data)].to_string())
```

Figure 25: Country Connections

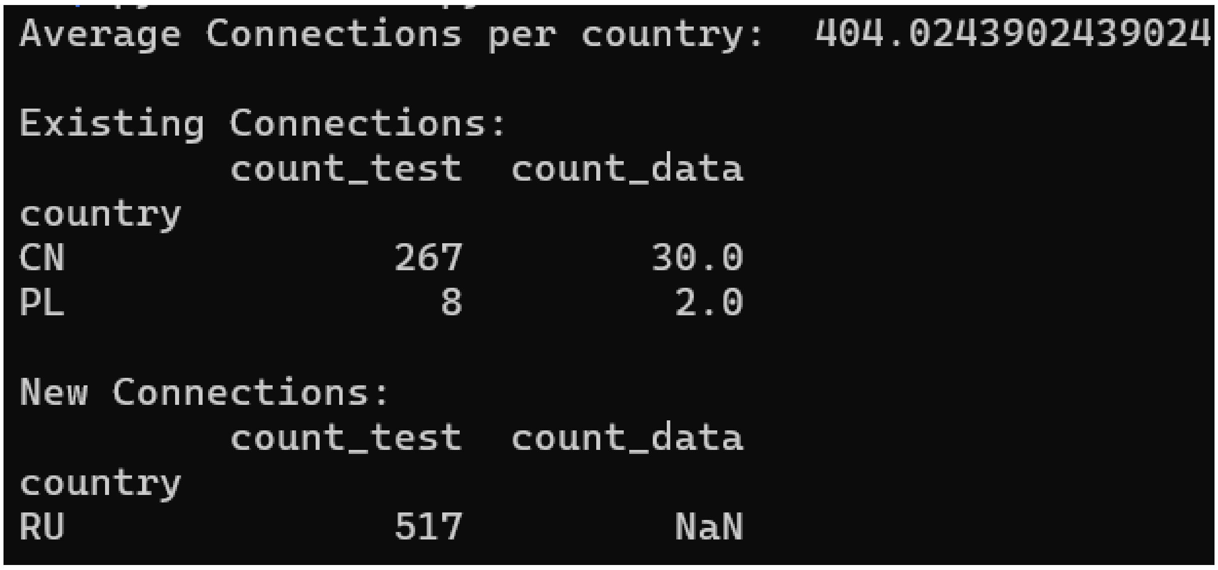
As seen above the rule triggers countries with 2 times the size of connections made on the data dataset, and only flags countries only present on the test data with connections above average.

Test

As we can 3 countries were detected from the values displayed.

For now, PL cannot be a dangerous country, even though it was triggered its number of connections is really low, in spite of increasing by 2.

However, CN and RU had a big increase in connections made which most likely are anomalous.



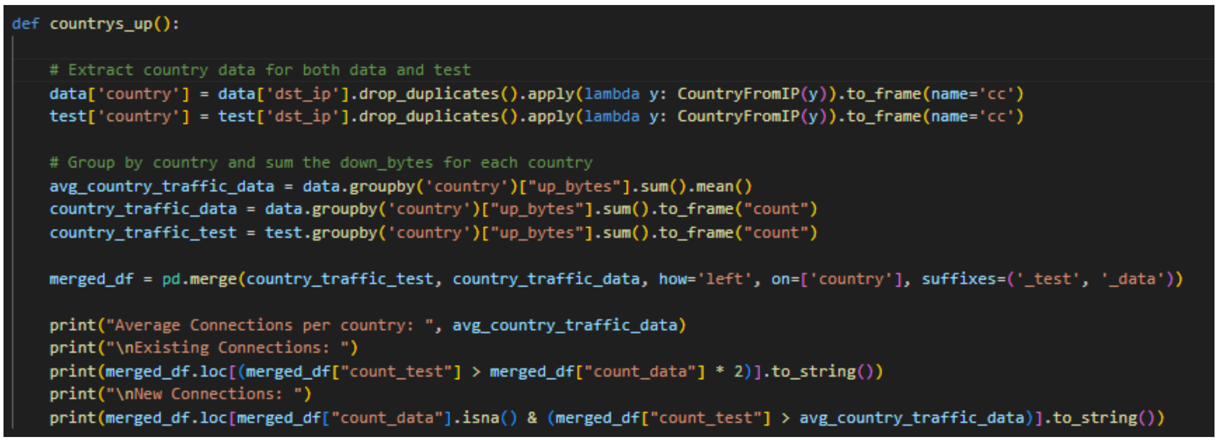
```
Average Connections per country:  404.0243902439024

Existing Connections:
          count_test  count_data
country
CN               267        30.0
PL                 8         2.0

New Connections:
          count_test  count_data
country
RU               517         NaN
```

Figure 26: Country Connections Result

3.1.2 countrys_up

The following rule is tailored to detecting abnormal amounts of uploaded data in each country.

This helps the organization detect if a country is trying to upload data from the organization which can indicate a data exfiltration attack.



```python
def countrys_up():

    # Extract country data for both data and test
    data['country'] = data['dst_ip'].drop_duplicates().apply(lambda y: CountryFromIP(y)).to_frame(name='cc')
    test['country'] = test['dst_ip'].drop_duplicates().apply(lambda y: CountryFromIP(y)).to_frame(name='cc')

    # Group by country and sum the down_bytes for each country
    avg_country_traffic_data = data.groupby('country')["up_bytes"].sum().mean()
    country_traffic_data = data.groupby('country')["up_bytes"].sum().to_frame("count")
    country_traffic_test = test.groupby('country')["up_bytes"].sum().to_frame("count")

    merged_df = pd.merge(country_traffic_test, country_traffic_data, how='left', on=['country'], suffixes=('_test', '_data'))

    print("Average Connections per country: ", avg_country_traffic_data)
    print("\nExisting Connections: ")
    print(merged_df.loc[(merged_df["count_test"] > merged_df["count_data"] * 2)].to_string())
    print("\nNew Connections: ")
    print(merged_df.loc[merged_df["count_data"].isna() & (merged_df["count_test"] > avg_country_traffic_data)].to_string())
```

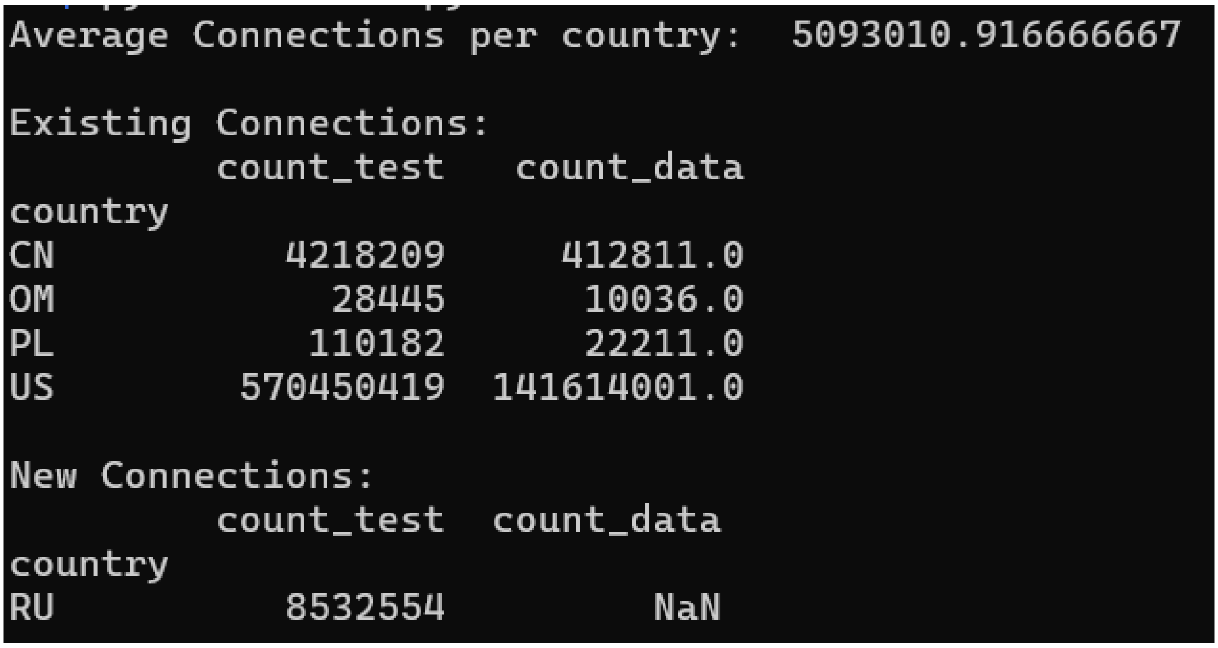
Figure 27: Country Uploads

As seen above the rule triggers countries with 2 times the amount of upload data made on the data dataset, and flags new countries with upload data above average.

Test

As we can see some countries such as PL and CN increased by 10 times and others like the US increased by 5 times. We can also see that RU is only 1.6 times above the average, however for a country which had no connections previously this makes us believe it can be doing some dubious activities.

Now we can also see that PL, even though in the previous rule countrys_conns, we couldn't conclude anything now we can since an increase of 10 times of uploads is very unlikely.



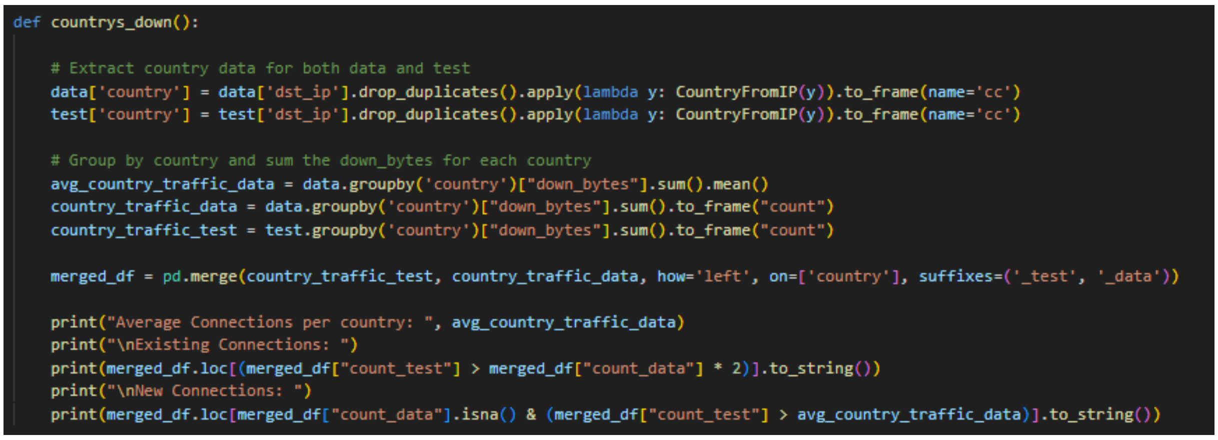
```
Average Connections per country:  5093010.916666667

Existing Connections:
         count_test   count_data
country
CN          4218209     412811.0
OM            28445      10036.0
PL           110182      22211.0
US        570450419  141614001.0

New Connections:
         count_test  count_data
country
RU          8532554         NaN
```

Figure 28: Country Uploads Result

3.1.3 countrys_down

The following rule aims to determine irregular amounts of Downloaded data in each country.

This helps the organization detect if a country is trying to download data to the organization which can indicate a C&C attack.



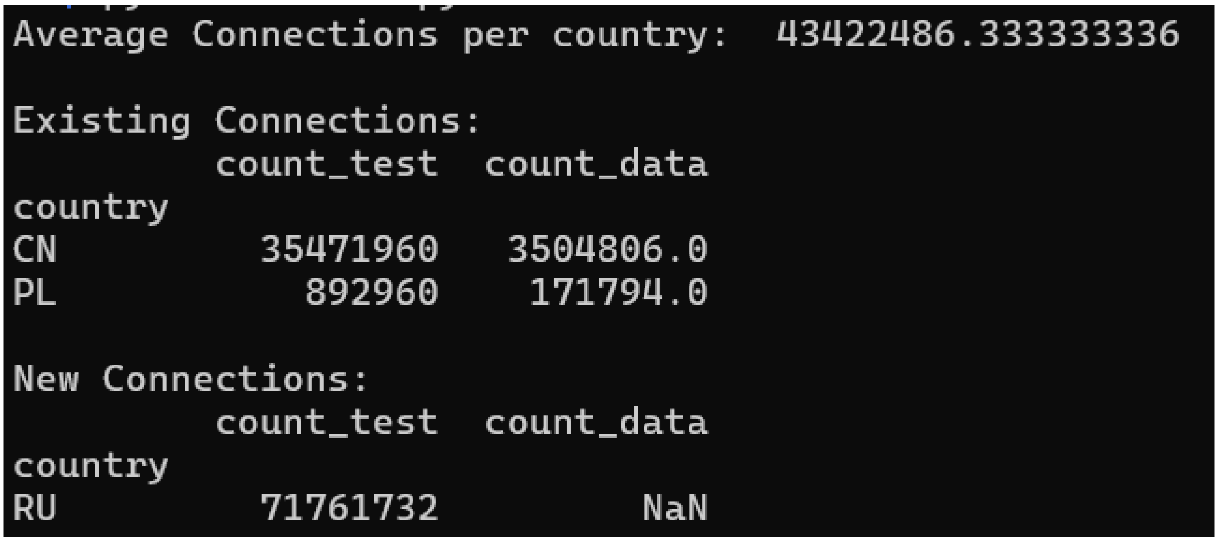
```python
def countrys_down():

    # Extract country data for both data and test
    data['country'] = data['dst_ip'].drop_duplicates().apply(lambda y: CountryFromIP(y)).to_frame(name='cc')
    test['country'] = test['dst_ip'].drop_duplicates().apply(lambda y: CountryFromIP(y)).to_frame(name='cc')

    # Group by country and sum the down_bytes for each country
    avg_country_traffic_data = data.groupby('country')["down_bytes"].sum().mean()
    country_traffic_data = data.groupby('country')["down_bytes"].sum().to_frame("count")
    country_traffic_test = test.groupby('country')["down_bytes"].sum().to_frame("count")

    merged_df = pd.merge(country_traffic_test, country_traffic_data, how='left', on=['country'], suffixes=('_test', '_data'))

    print("Average Connections per country: ", avg_country_traffic_data)
    print("\nExisting Connections: ")
    print(merged_df.loc[(merged_df["count_test"] > merged_df["count_data"] * 2)].to_string())
    print("\nNew Connections: ")
    print(merged_df.loc[merged_df["count_data"].isna() & (merged_df["count_test"] > avg_country_traffic_data)].to_string())
```

Figure 29: Country Downloads

As seen above the rule triggers countries with 2 times the amount of downloaded data made on the data dataset, and flags new countries with upload data above average.

Test

With these results, we can conclude once again that CN and PL have strange behavior since their traffic went close to 10 times higher and RU increased by 8 times.



```
Average Connections per country:  43422486.333333336

Existing Connections:
         count_test  count_data
country
CN         35471960   3504806.0
PL           892960    171794.0

New Connections:
         count_test  count_data
country
RU         71761732         NaN
```

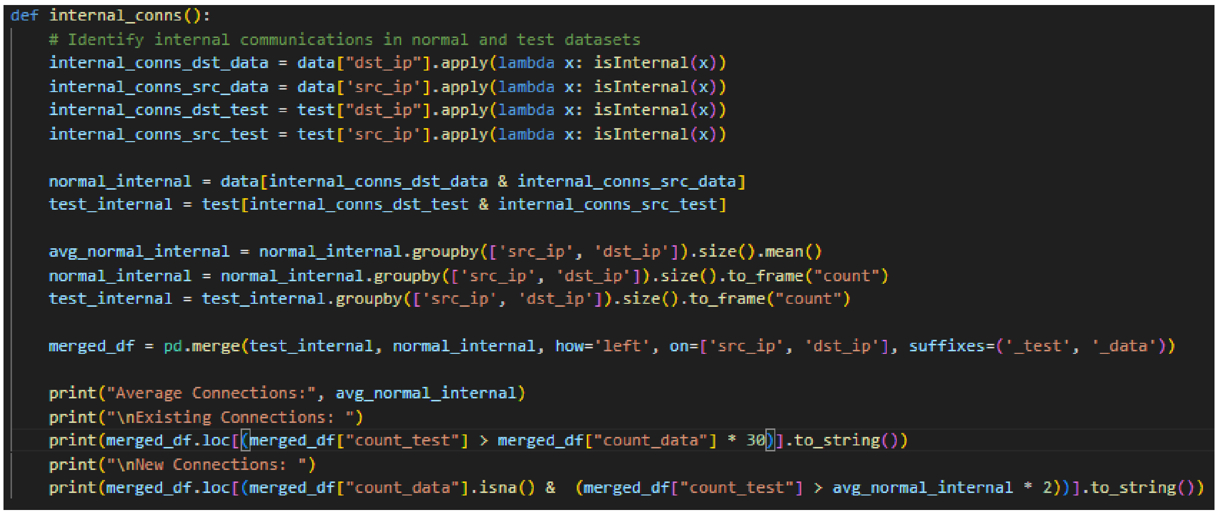
Figure 30: Country Downloads Result

3.2 Network Connections/Requests

3.2.1 internal_conns

The following rule will try to detect connections between devices associated with a botnet.

This rule is crucial as it detects if a source IP attempts to establish connections with the same device more frequently than usual. If only the source IP were considered, as demonstrated in internal_ip_conns, a botnet could decrease its connections to a specific IP while increasing connections to others, effectively masking its activity.



```
def internal_conns():
    # Identify internal communications in normal and test datasets
    internal_conns_dst_data = data["dst_ip"].apply(lambda x: isInternal(x))
    internal_conns_src_data = data['src_ip'].apply(lambda x: isInternal(x))
    internal_conns_dst_test = test["dst_ip"].apply(lambda x: isInternal(x))
    internal_conns_src_test = test['src_ip'].apply(lambda x: isInternal(x))

    normal_internal = data[internal_conns_dst_data & internal_conns_src_data]
    test_internal = test[internal_conns_dst_test & internal_conns_src_test]

    avg_normal_internal = normal_internal.groupby(['src_ip', 'dst_ip']).size().mean()
    normal_internal = normal_internal.groupby(['src_ip', 'dst_ip']).size().to_frame("count")
    test_internal = test_internal.groupby(['src_ip', 'dst_ip']).size().to_frame("count")

    merged_df = pd.merge(test_internal, normal_internal, how='left', on=['src_ip', 'dst_ip'], suffixes=('_test', '_data'))

    print("Average Connections:", avg_normal_internal)
    print("\nExisting Connections: ")
    print(merged_df.loc[(merged_df["count_test"] > merged_df["count_data"] * 30)].to_string())
    print("\nNew Connections: ")
    print(merged_df.loc[(merged_df["count_data"].isna() &  (merged_df["count_test"] > avg_normal_internal * 2))].to_string())
```

Figure 31: Internal Connections

To begin it filters traffic which have an internal source and destination IP. Then it creates a group for every source and destination IPs and calculates the number of requests made between them. Additionally, it computes the average requests between all groups.

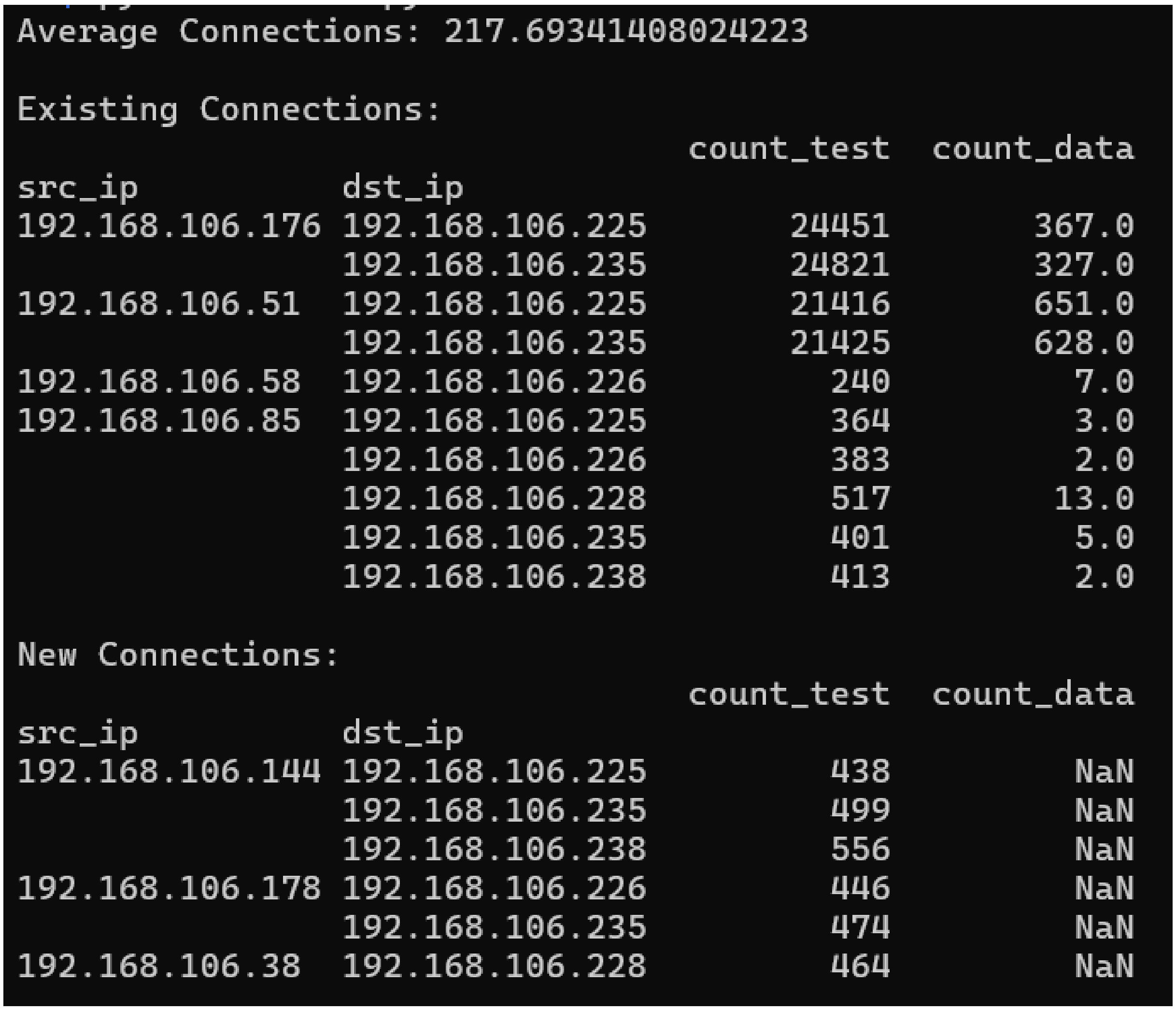
As seen above if a connection is present in both datasets and the test dataset shows a connection 30 times greater than the data dataset, that IP is flagged.

Groups exclusively in the test dataset are compared to the average number of requests per group, and if this number is twice as high, the device is flagged as suspicious.

Testing

As seen below multiple devices were flagged.

It's important to note that 4 of these groups which have a source IP of 192.168.106.176 and 192.168.106.51 have connections 10 times bigger than their normal behavior which makes us conclude that some undesired activities are being conducted on these IPs.

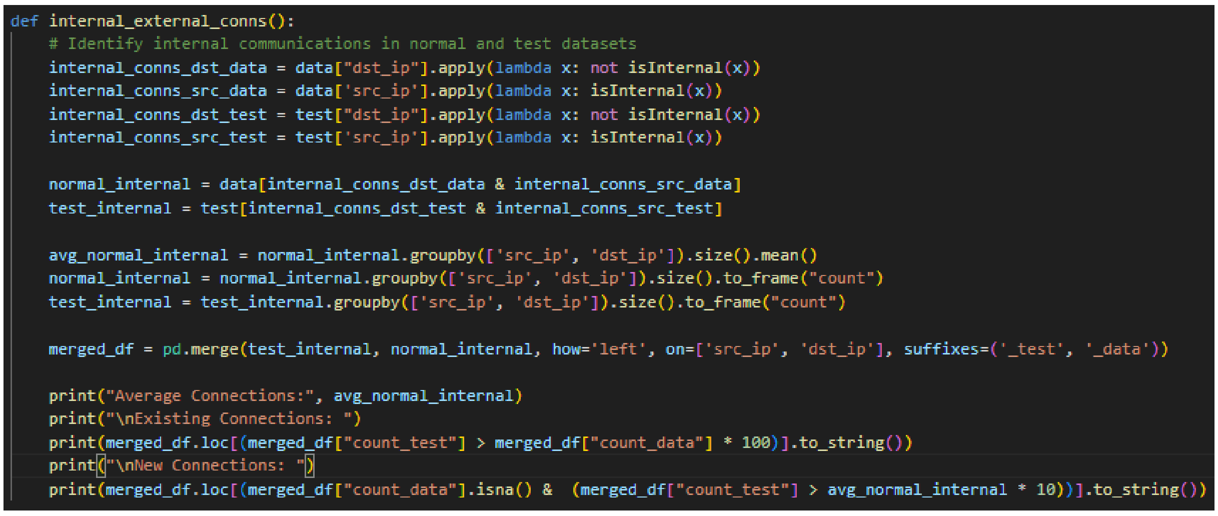


```
Average Connections: 217.69341408024223

Existing Connections:
                                  count_test  count_data
src_ip          dst_ip
192.168.106.176 192.168.106.225        24451       367.0
                192.168.106.235        24821       327.0
192.168.106.51  192.168.106.225        21416       651.0
                192.168.106.235        21425       628.0
192.168.106.58  192.168.106.226          240         7.0
192.168.106.85  192.168.106.225          364         3.0
                192.168.106.226          383         2.0
                192.168.106.228          517        13.0
                192.168.106.235          401         5.0
                192.168.106.238          413         2.0

New Connections:
                                  count_test  count_data
src_ip          dst_ip
192.168.106.144 192.168.106.225          438         NaN
                192.168.106.235          499         NaN
                192.168.106.238          556         NaN
192.168.106.178 192.168.106.226          446         NaN
                192.168.106.235          474         NaN
192.168.106.38  192.168.106.228          464         NaN
```

Figure 32: Internal Connections Results

3.2.2 internal_external_conns

The following rule works similarly to internal_conns, However, the key difference relies on the fact that groups are made between internal source IPs with external destination IPs, trying to detect devices communicating with external IPs for a possible **Data Exfiltration attack**.



```python
def internal_external_conns():
    # Identify internal communications in normal and test datasets
    internal_conns_dst_data = data["dst_ip"].apply(lambda x: not isInternal(x))
    internal_conns_src_data = data['src_ip'].apply(lambda x: isInternal(x))
    internal_conns_dst_test = test["dst_ip"].apply(lambda x: not isInternal(x))
    internal_conns_src_test = test['src_ip'].apply(lambda x: isInternal(x))

    normal_internal = data[internal_conns_dst_data & internal_conns_src_data]
    test_internal = test[internal_conns_dst_test & internal_conns_src_test]

    avg_normal_internal = normal_internal.groupby(['src_ip', 'dst_ip']).size().mean()
    normal_internal = normal_internal.groupby(['src_ip', 'dst_ip']).size().to_frame("count")
    test_internal = test_internal.groupby(['src_ip', 'dst_ip']).size().to_frame("count")

    merged_df = pd.merge(test_internal, normal_internal, how='left', on=['src_ip', 'dst_ip'], suffixes=('_test', '_data'))

    print("Average Connections:", avg_normal_internal)
    print("\nExisting Connections: ")
    print(merged_df.loc[(merged_df["count_test"] > merged_df["count_data"] * 100)].to_string())
    print("\nNew Connections: ")
    print(merged_df.loc[(merged_df["count_data"].isna() &  (merged_df["count_test"] > avg_normal_internal * 10))].to_string())
```

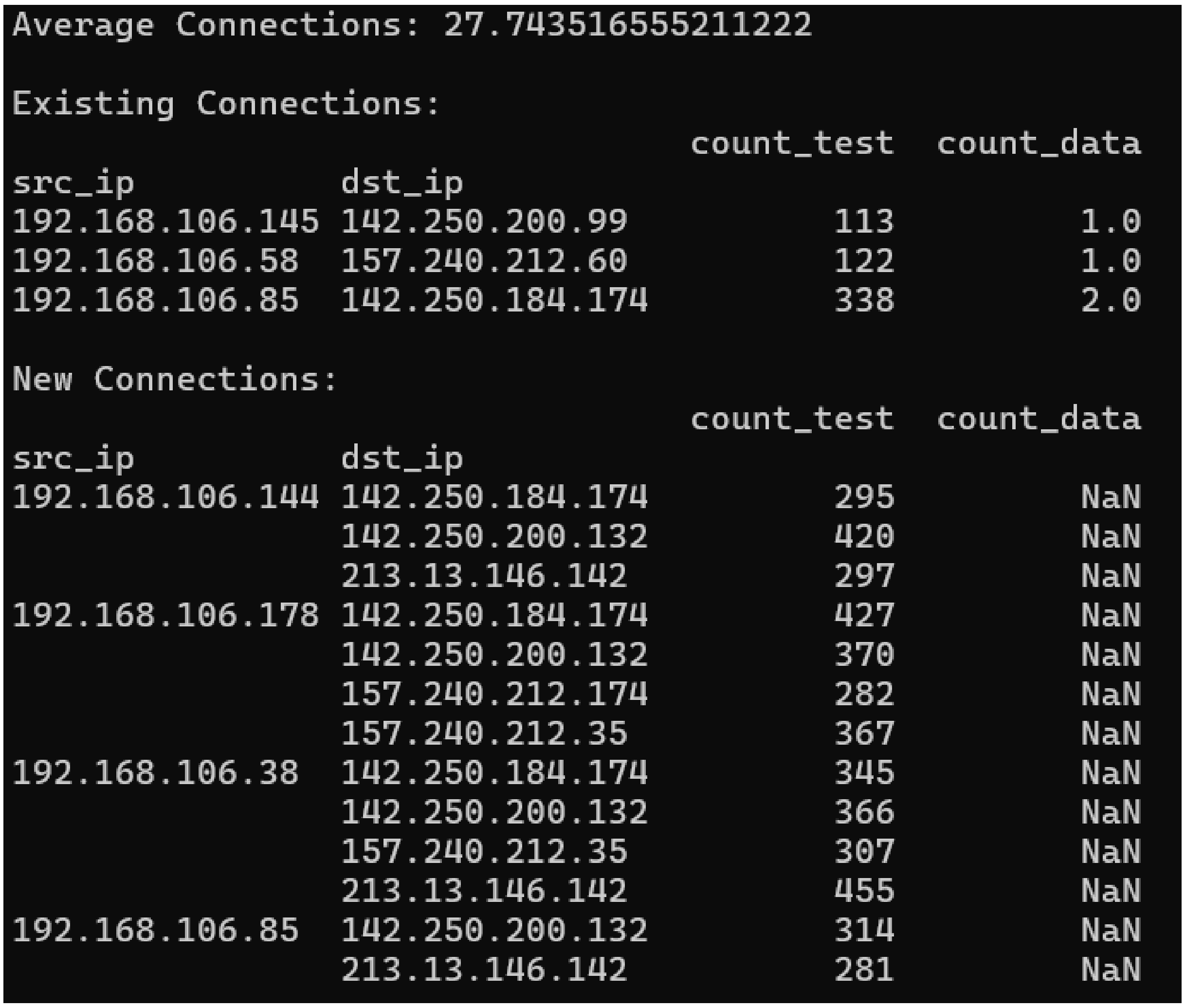
Figure 33: Internal to external Connections

As seen above existing groups need to have 100 times more connections to trigger a flag, and for non existing the average needs to be 10 times to flag the internal IP.

Test

From the results shown below we can see that the number of connections for these devices increased by 10 or even 20 times the normal amount.

With this info, we can also speculate some destion IPs with malicious intent, the 142.250.184.174, 142.250.184.132, and 157.250.184.174 IPs are all present in multiple flagged connections making us believe they are an important piece on the attack being conducted.



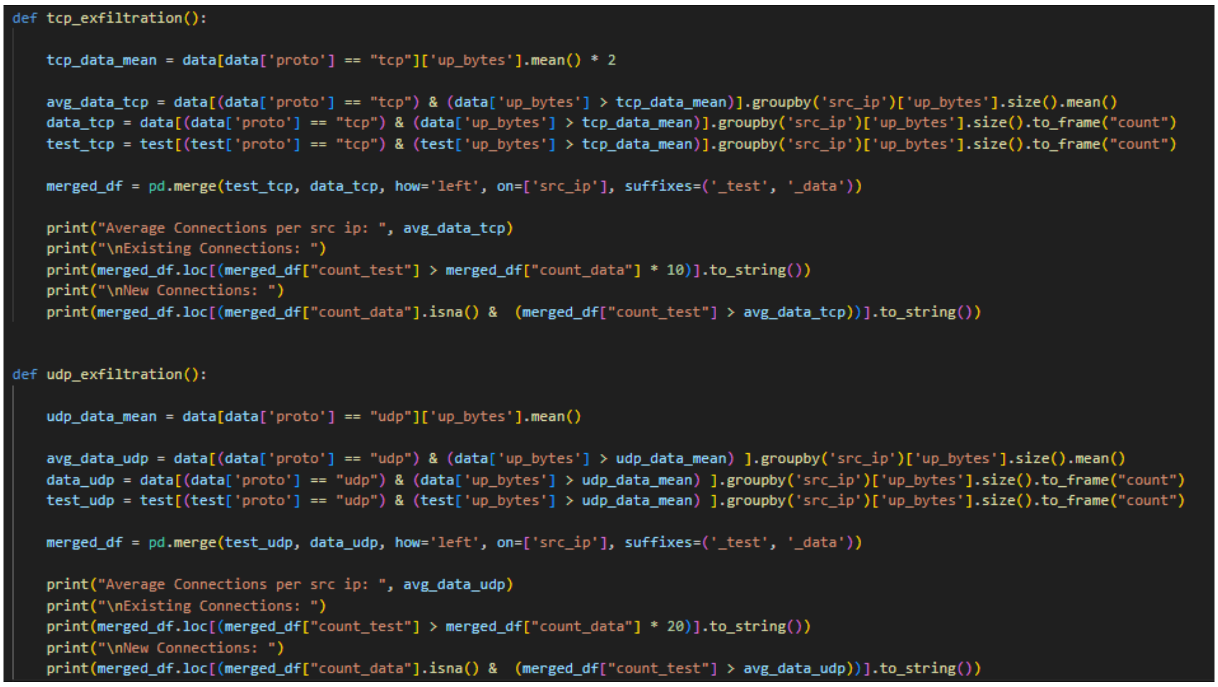
```
Average Connections: 27.743516555211222

Existing Connections:
                                 count_test  count_data
src_ip          dst_ip
192.168.106.145 142.250.200.99          113         1.0
192.168.106.58  157.240.212.60          122         1.0
192.168.106.85  142.250.184.174         338         2.0

New Connections:
                                 count_test  count_data
src_ip          dst_ip
192.168.106.144 142.250.184.174         295         NaN
                142.250.200.132         420         NaN
                213.13.146.142          297         NaN
192.168.106.178 142.250.184.174         427         NaN
                142.250.200.132         370         NaN
                157.240.212.174         282         NaN
                157.240.212.35          367         NaN
192.168.106.38  142.250.184.174         345         NaN
                142.250.200.132         366         NaN
                157.240.212.35          307         NaN
                213.13.146.142          455         NaN
192.168.106.85  142.250.200.132         314         NaN
                213.13.146.142          281         NaN
```

Figure 34: Internal to external Connections Results

3.2.3 tcp_exfiltration and udp_exfiltration

The following Rules compute the number of upload bytes done by the HTTPS(TCP on port 443) and DNS(UDP on port 43) protocols.

The objective of this rule is to detect abnormal amounts of uploaded data which can indicate data exfiltration.



```python
def tcp_exfiltration():

    tcp_data_mean = data[data['proto'] == "tcp"]['up_bytes'].mean() * 2

    avg_data_tcp = data[(data['proto'] == "tcp") & (data['up_bytes'] > tcp_data_mean)].groupby('src_ip')['up_bytes'].size().mean()
    data_tcp = data[(data['proto'] == "tcp") & (data['up_bytes'] > tcp_data_mean)].groupby('src_ip')['up_bytes'].size().to_frame("count")
    test_tcp = test[(test['proto'] == "tcp") & (test['up_bytes'] > tcp_data_mean)].groupby('src_ip')['up_bytes'].size().to_frame("count")

    merged_df = pd.merge(test_tcp, data_tcp, how='left', on=['src_ip'], suffixes=('_test', '_data'))

    print("Average Connections per src ip: ", avg_data_tcp)
    print("\nExisting Connections: ")
    print(merged_df.loc[(merged_df["count_test"] > merged_df["count_data"] * 10)].to_string())
    print("\nNew Connections: ")
    print(merged_df.loc[(merged_df["count_data"].isna() &  (merged_df["count_test"] > avg_data_tcp))].to_string())


def udp_exfiltration():

    udp_data_mean = data[data['proto'] == "udp"]['up_bytes'].mean()

    avg_data_udp = data[(data['proto'] == "udp") & (data['up_bytes'] > udp_data_mean) ].groupby('src_ip')['up_bytes'].size().mean()
    data_udp = data[(data['proto'] == "udp") & (data['up_bytes'] > udp_data_mean) ].groupby('src_ip')['up_bytes'].size().to_frame("count")
    test_udp = test[(test['proto'] == "udp") & (test['up_bytes'] > udp_data_mean) ].groupby('src_ip')['up_bytes'].size().to_frame("count")

    merged_df = pd.merge(test_udp, data_udp, how='left', on=['src_ip'], suffixes=('_test', '_data'))

    print("Average Connections per src ip: ", avg_data_udp)
    print("\nExisting Connections: ")
    print(merged_df.loc[(merged_df["count_test"] > merged_df["count_data"] * 20)].to_string())
    print("\nNew Connections: ")
    print(merged_df.loc[(merged_df["count_data"].isna() &  (merged_df["count_test"] > avg_data_udp))].to_string())
```

Figure 35: Data Exfiltration

As seen on above, these rules begin by computing the mean of uploaded bytes for TCP/UDP packets in the data dataset and then doubling this value.

Subsequently, it determines the average number of TCP connections with uploaded bytes exceeding the doubled mean in both the data and test datasets, grouping them by source IP address, for the UDP it only calculates the average number of connections exceeding the mean.

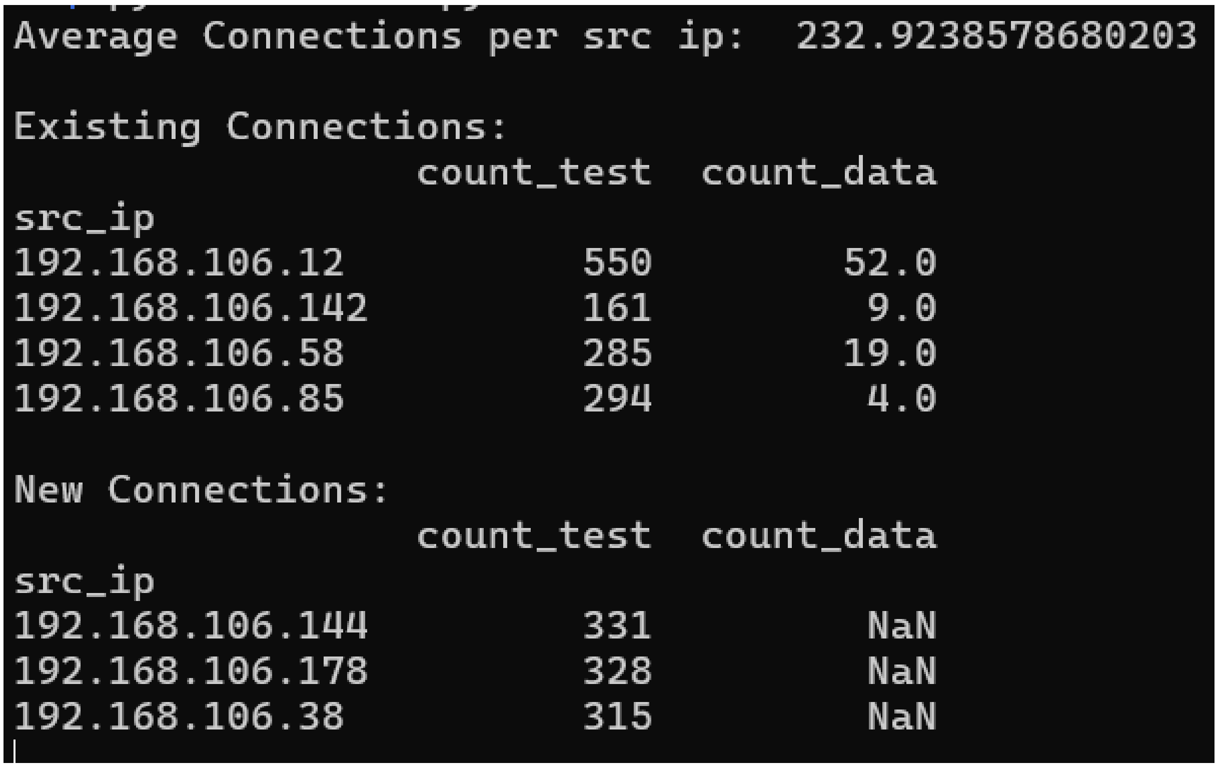
Finally, these functions compare connections from the data dataset that exceed the doubled mean for the TCP and only above the mean for the UDP, of uploaded bytes with those in the test dataset. If a source IP in the test dataset has connections exceeding ten times for the TCP and 20 times for the UDP the corresponding connections in the data dataset are flagged.

Additionally, new connections that exist only in the test dataset are flagged if their count surpasses the average number of connections that exceeded the doubled mean in the data dataset.

Test

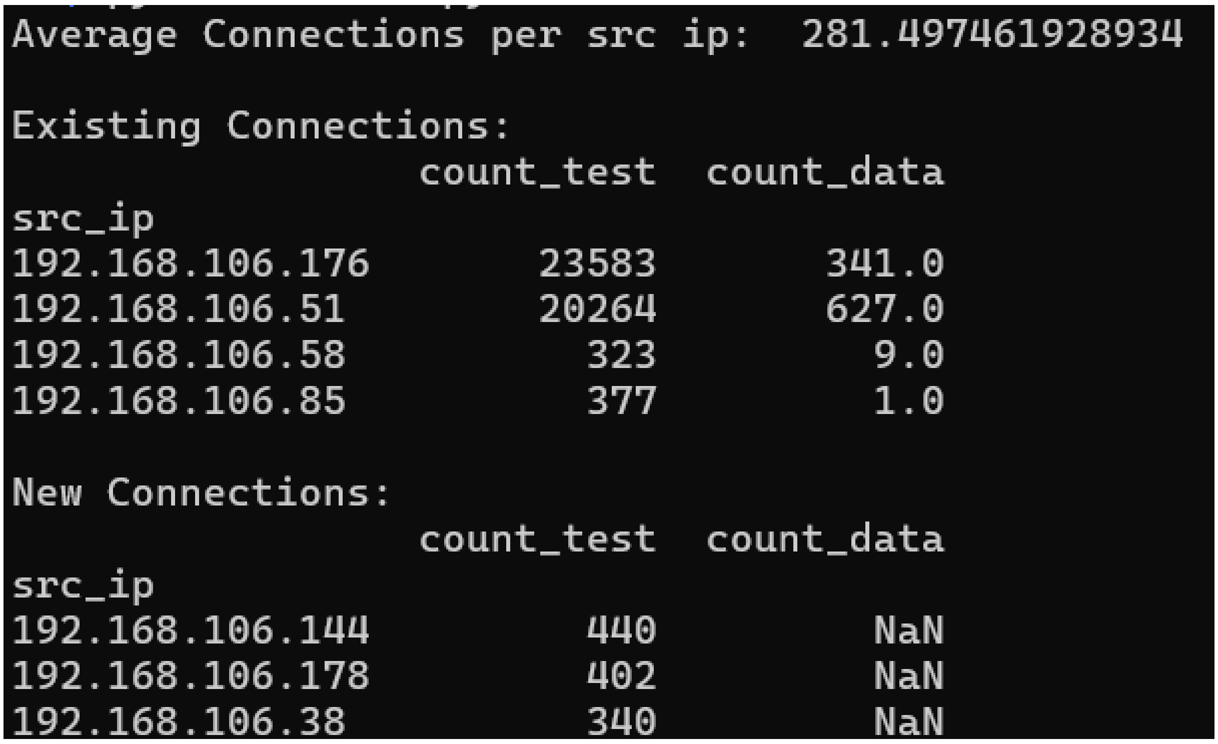
As we can see on the results below, there were multiple IPs flagged, we should take special attention to the IPs 192.168.106.176 and 192.168.106.51 as they have almost 70 times more connections that surpass the average of normal uploads.

Also by looking into both flagged results, we can see that they have equivalent IPS, further indicating that these IPs that these IPs are likely engaging in data exfiltration attacks, using both protocols.



```
Average Connections per src ip:  232.9238578680203

Existing Connections:
                 count_test  count_data
src_ip
192.168.106.12          550        52.0
192.168.106.142         161         9.0
192.168.106.58          285        19.0
192.168.106.85          294         4.0

New Connections:
                 count_test  count_data
src_ip
192.168.106.144         331         NaN
192.168.106.178         328         NaN
192.168.106.38          315         NaN
```

Figure 36: TCP Exfiltration Results

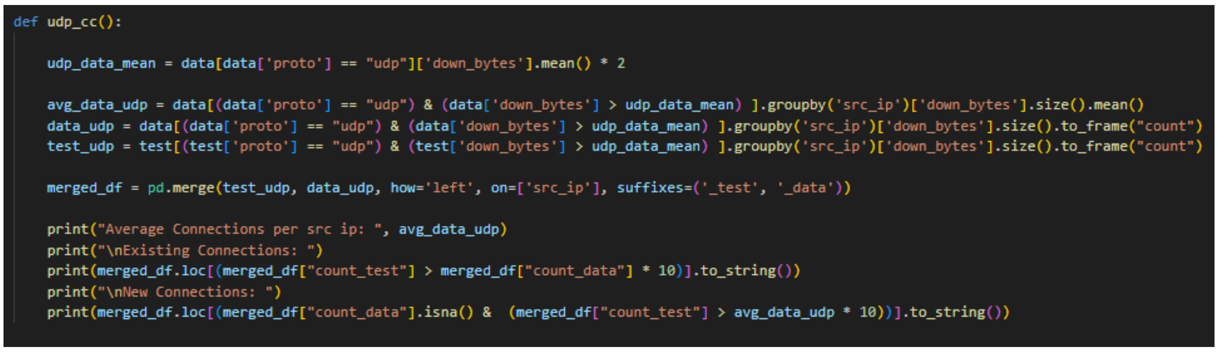


```
Average Connections per src ip:  281.497461928934

Existing Connections:
                 count_test  count_data
src_ip
192.168.106.176       23583       341.0
192.168.106.51        20264       627.0
192.168.106.58          323         9.0
192.168.106.85          377         1.0

New Connections:
                 count_test  count_data
src_ip
192.168.106.144         440         NaN
192.168.106.178         402         NaN
192.168.106.38          340         NaN
```

Figure 37: UDP Exfiltration Results

3.2.4 udp_cc

The following rule used similar concepts as the previous rule tcp_exfiltration and udp_exfiltration, however instead of observing the uploaded data it observed for downloaded data trying to detect possible C&C attacks.

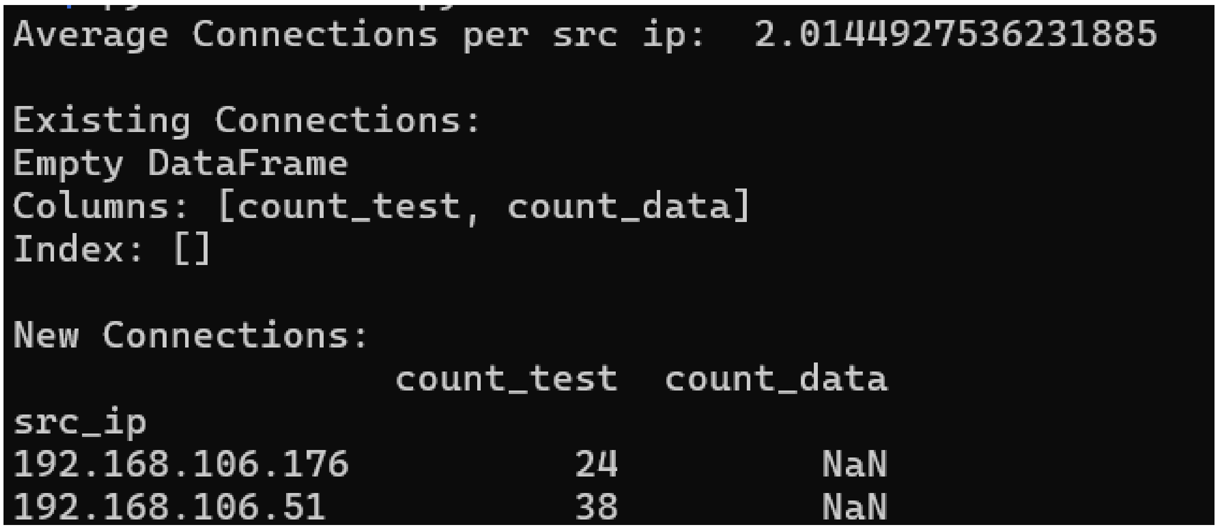


```python
def udp_cc():

    udp_data_mean = data[data['proto'] == "udp"]['down_bytes'].mean() * 2

    avg_data_udp = data[(data['proto'] == "udp") & (data['down_bytes'] > udp_data_mean) ].groupby('src_ip')['down_bytes'].size().mean()
    data_udp = data[(data['proto'] == "udp") & (data['down_bytes'] > udp_data_mean) ].groupby('src_ip')['down_bytes'].size().to_frame("count")
    test_udp = test[(test['proto'] == "udp") & (test['down_bytes'] > udp_data_mean) ].groupby('src_ip')['down_bytes'].size().to_frame("count")

    merged_df = pd.merge(test_udp, data_udp, how='left', on=['src_ip'], suffixes=('_test', '_data'))

    print("Average Connections per src ip: ", avg_data_udp)
    print("\nExisting Connections: ")
    print(merged_df.loc[(merged_df["count_test"] > merged_df["count_data"] * 10)].to_string())
    print("\nNew Connections: ")
    print(merged_df.loc[(merged_df["count_data"].isna() &  (merged_df["count_test"] > avg_data_udp * 10))].to_string())
```

Figure 38: UDP C&C

Test

As shown in the results 2 IPs had an abnormal amount of connections surpassing the normal downloaded data.



```
Average Connections per src ip:  2.0144927536231885

Existing Connections:
Empty DataFrame
Columns: [count_test, count_data]
Index: []

New Connections:
                 count_test  count_data
src_ip
192.168.106.176          24         NaN
192.168.106.51           38         NaN
```

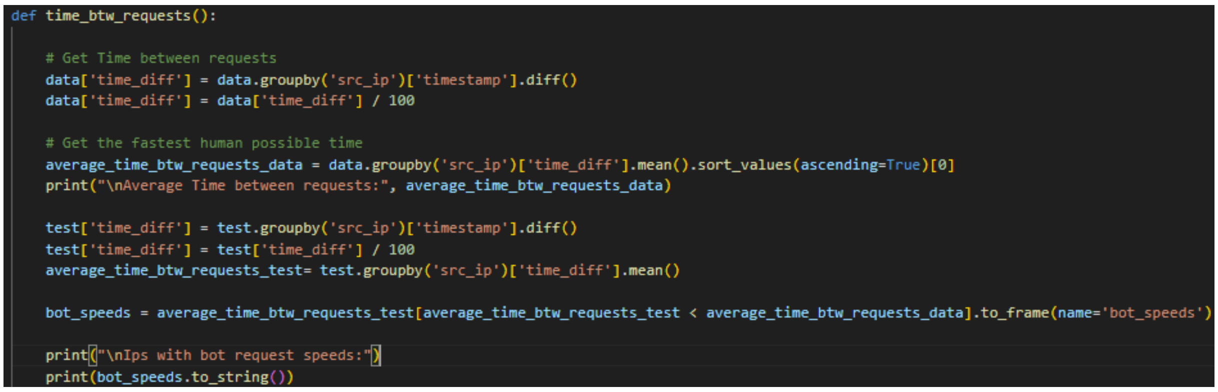
Figure 39: UDP C&C Results

3.3 Individual IP Behaviors

3.3.1 time_btw_requests

The following rule aims to identify botnet activities by trying to detect requests made with unusual time between them.

It involves detecting user requests, comparing their timing with the fastest request time in the **normal dataset**, and flagging any devices that make requests at quicker intervals on the **test dataset**.

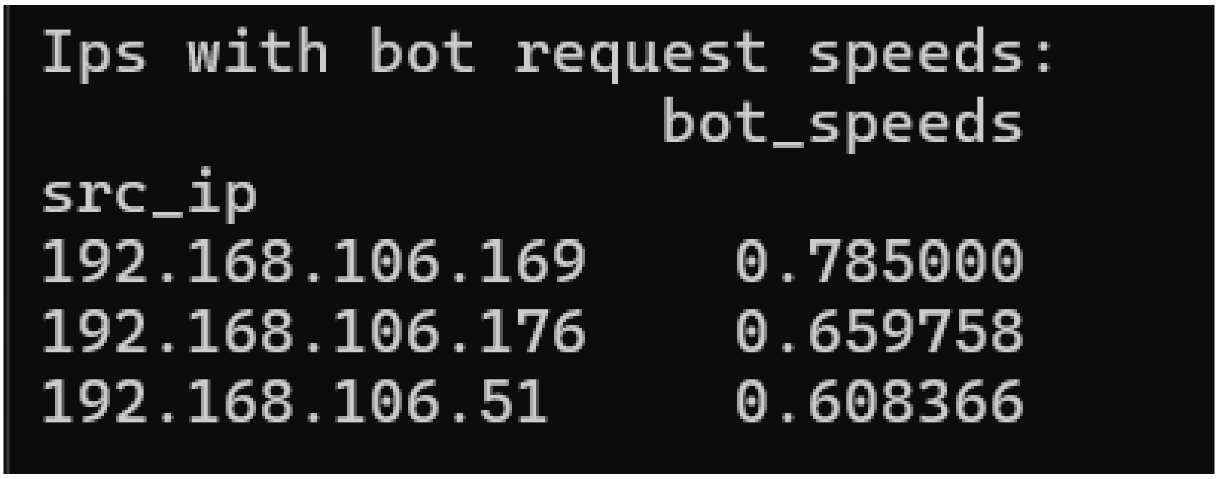


```python
def time_btw_requests():

    # Get Time between requests
    data['time_diff'] = data.groupby('src_ip')['timestamp'].diff()
    data['time_diff'] = data['time_diff'] / 100

    # Get the fastest human possible time
    average_time_btw_requests_data = data.groupby('src_ip')['time_diff'].mean().sort_values(ascending=True)[0]
    print("\nAverage Time between requests:", average_time_btw_requests_data)

    test['time_diff'] = test.groupby('src_ip')['timestamp'].diff()
    test['time_diff'] = test['time_diff'] / 100
    average_time_btw_requests_test= test.groupby('src_ip')['time_diff'].mean()

    bot_speeds = average_time_btw_requests_test[average_time_btw_requests_test < average_time_btw_requests_data].to_frame(name='bot_speeds')

    print("\nIps with bot request speeds:")
    print(bot_speeds.to_string())
```

Figure 40: Time between requests

Test

As seen below 3 devices were detected, the average interval between normal requests was 3 seconds, while these devices were submitting requests at intervals less than 1 second. Which makes us conclude that this devices are bots.



```
Ips with bot request speeds:
                 bot_speeds
src_ip
192.168.106.169    0.785000
192.168.106.176    0.659758
192.168.106.51     0.608366
```

Figure 41: Time between requests Results

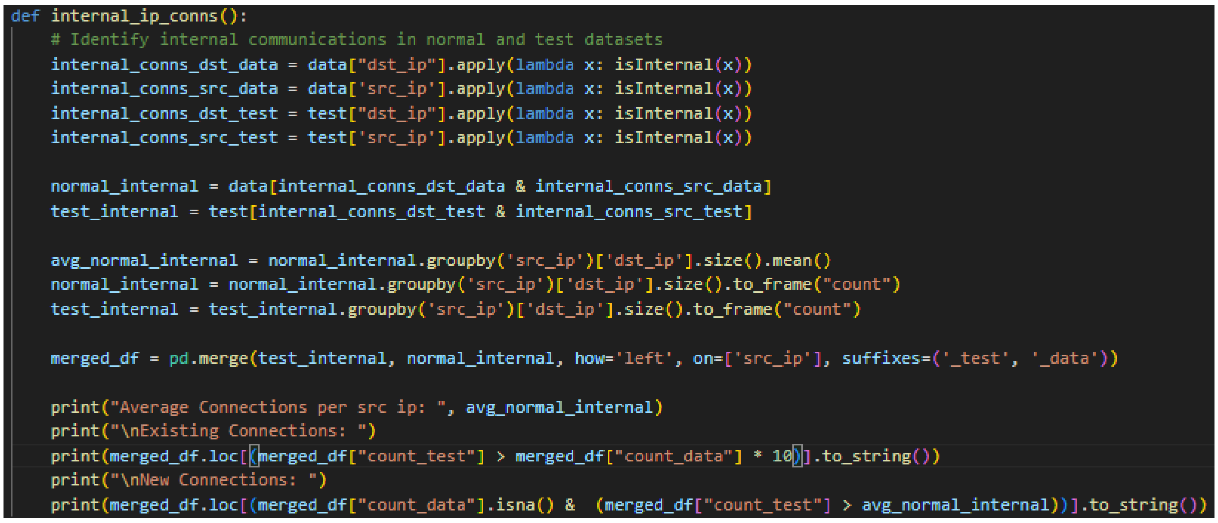
3.3.2 internal_ip_conns

This rule works similarly to the internal_conns rule the core difference is that the connections computed are solely based on the source IP and its connections between any internal destination IP. Which has to have 10 times more connections than the data and only be above average for the IPs that don't exist on the data dataset.

Unlike the previous rule, which focuses on detecting connections between specific source and destination IP pairs, this rule provides an overview of all connections involving a specific

IP address.

This broader view can detect connections not increasing for a single group, source-destination pair, but between a greater range of destination IPs.

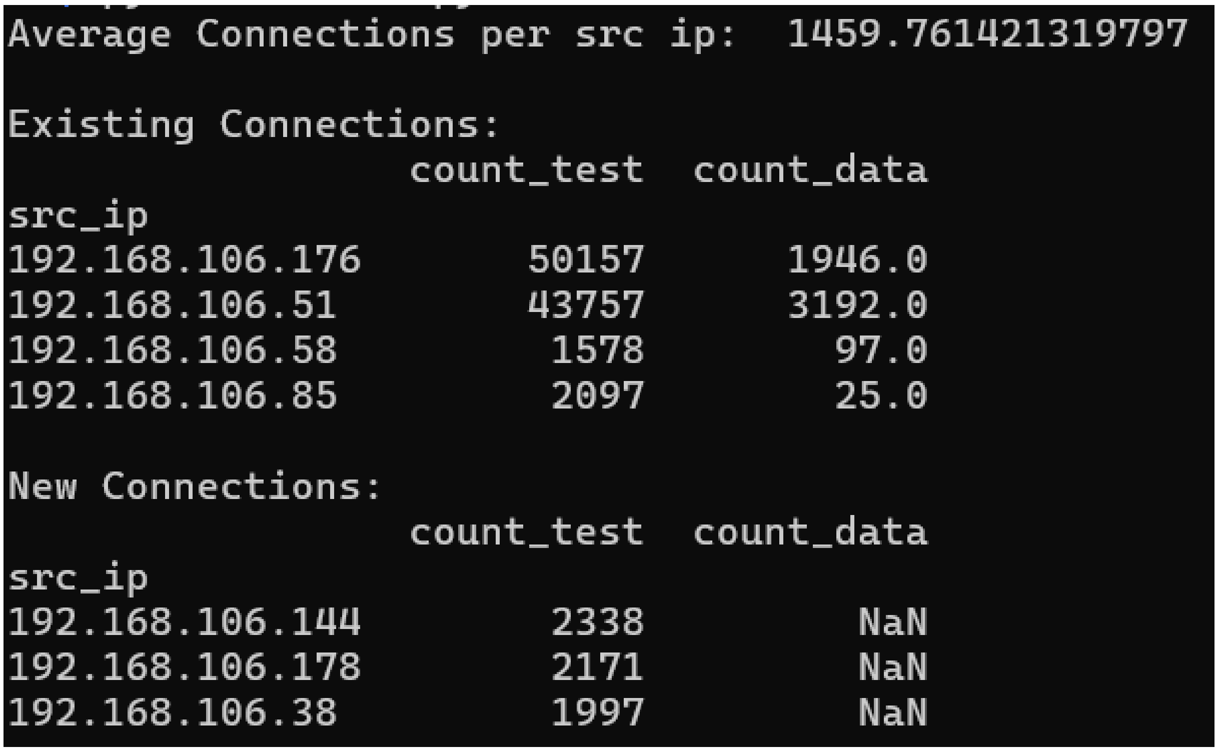


```
def internal_ip_conns():
    # Identify internal communications in normal and test datasets
    internal_conns_dst_data = data["dst_ip"].apply(lambda x: isInternal(x))
    internal_conns_src_data = data['src_ip'].apply(lambda x: isInternal(x))
    internal_conns_dst_test = test["dst_ip"].apply(lambda x: isInternal(x))
    internal_conns_src_test = test['src_ip'].apply(lambda x: isInternal(x))

    normal_internal = data[internal_conns_dst_data & internal_conns_src_data]
    test_internal = test[internal_conns_dst_test & internal_conns_src_test]

    avg_normal_internal = normal_internal.groupby('src_ip')['dst_ip'].size().mean()
    normal_internal = normal_internal.groupby('src_ip')['dst_ip'].size().to_frame("count")
    test_internal = test_internal.groupby('src_ip')['dst_ip'].size().to_frame("count")

    merged_df = pd.merge(test_internal, normal_internal, how='left', on=['src_ip'], suffixes=('_test', '_data'))

    print("Average Connections per src ip: ", avg_normal_internal)
    print("\nExisting Connections: ")
    print(merged_df.loc[(merged_df["count_test"] > merged_df["count_data"] * 10)].to_string())
    print("\nNew Connections: ")
    print(merged_df.loc[(merged_df["count_data"].isna() &  (merged_df["count_test"] > avg_normal_internal))].to_string())
```

Figure 42: Internal IP connections

Test

On this specific dataset, the devices found were equal to the ones of the previous rule internal_conns, which improves the probability of this being part of a botnet attack.

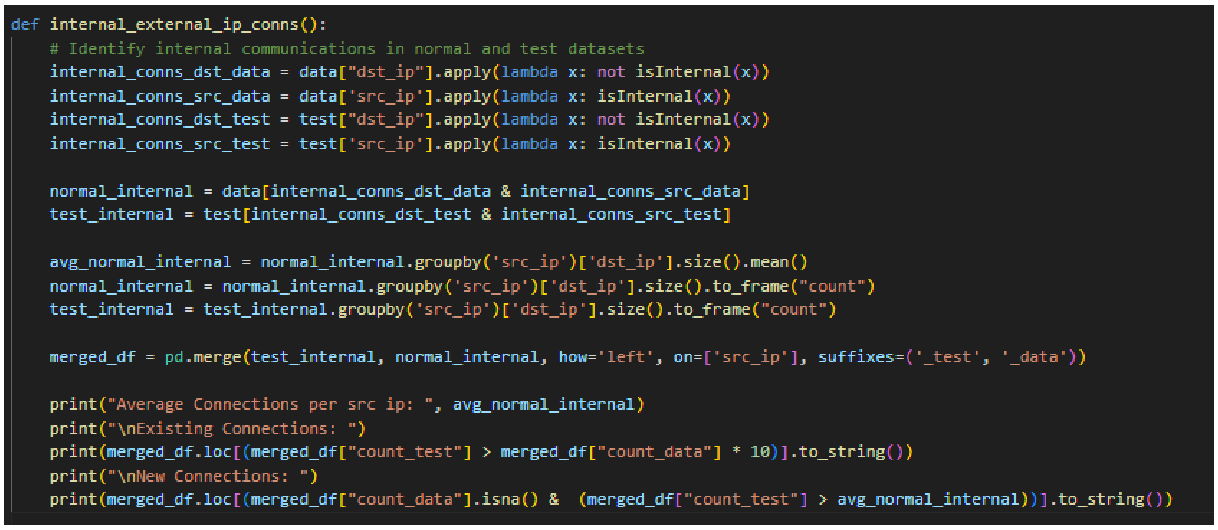


```
Average Connections per src ip:  1459.761421319797

Existing Connections:
                  count_test  count_data
src_ip
192.168.106.176        50157      1946.0
192.168.106.51         43757      3192.0
192.168.106.58          1578        97.0
192.168.106.85          2097        25.0

New Connections:
                  count_test  count_data
src_ip
192.168.106.144         2338         NaN
192.168.106.178         2171         NaN
192.168.106.38          1997         NaN
```

Figure 43: Internal IP connections Results

3.3.3 internal_external_ip_conns

The following rule has the same objective as the internal_external_conns, this will try to see unnatural increase in the connections between internal devices to external devices.



```python
def internal_external_ip_conns():
    # Identify internal communications in normal and test datasets
    internal_conns_dst_data = data["dst_ip"].apply(lambda x: not isInternal(x))
    internal_conns_src_data = data['src_ip'].apply(lambda x: isInternal(x))
    internal_conns_dst_test = test["dst_ip"].apply(lambda x: not isInternal(x))
    internal_conns_src_test = test['src_ip'].apply(lambda x: isInternal(x))

    normal_internal = data[internal_conns_dst_data & internal_conns_src_data]
    test_internal = test[internal_conns_dst_test & internal_conns_src_test]

    avg_normal_internal = normal_internal.groupby('src_ip')['dst_ip'].size().mean()
    normal_internal = normal_internal.groupby('src_ip')['dst_ip'].size().to_frame("count")
    test_internal = test_internal.groupby('src_ip')['dst_ip'].size().to_frame("count")

    merged_df = pd.merge(test_internal, normal_internal, how='left', on=['src_ip'], suffixes=('_test', '_data'))

    print("Average Connections per src ip: ", avg_normal_internal)
    print("\nExisting Connections: ")
    print(merged_df.loc[(merged_df["count_test"] > merged_df["count_data"] * 10)].to_string())
    print("\nNew Connections: ")
    print(merged_df.loc[(merged_df["count_data"].isna() &  (merged_df["count_test"] > avg_normal_internal))].to_string())
```

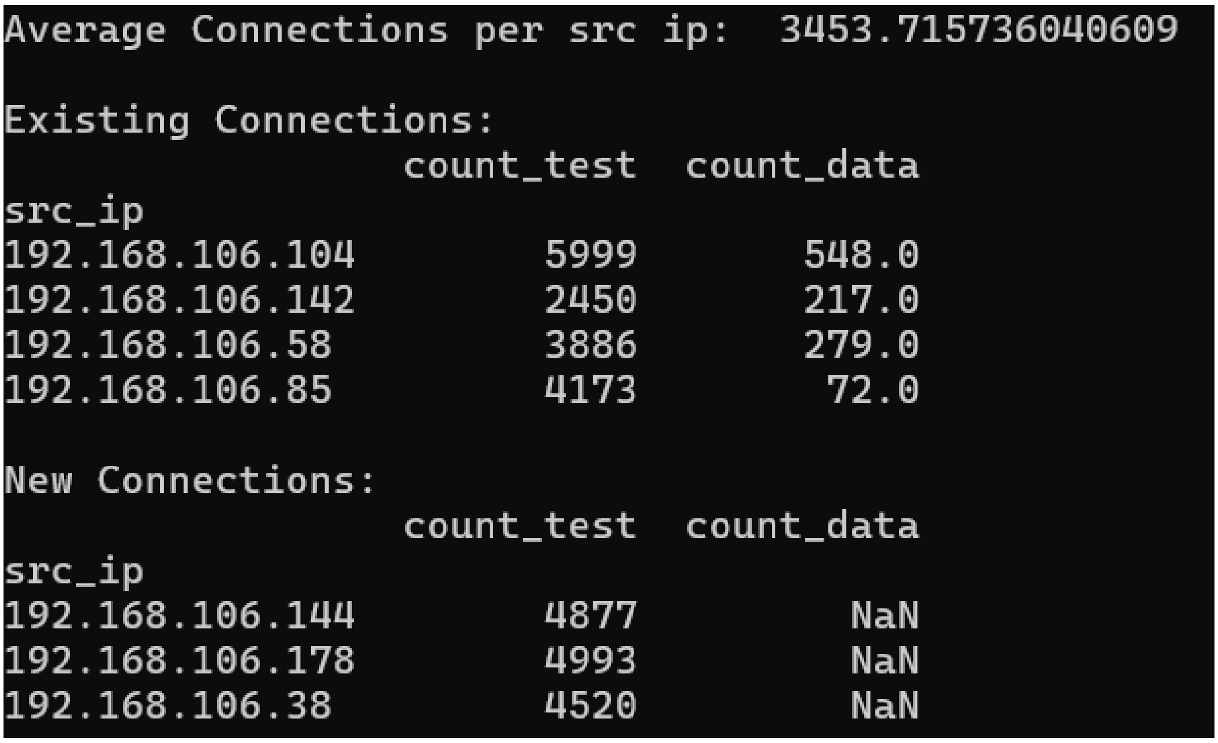
Figure 44: Internal to external IP connections

This function begins by filtering all connections made to external IP addresses. It then calculates the average number of connections that each source IP makes to external devices. Additionally, it computes the number of connections each device has in both the data dataset and the test dataset.

Finally, the function compares the behaviors of non-anomalous source IPs with those in the test dataset. If the test dataset shows an increase of 10 times in connections compared to the data dataset, it flags this behavior. Moreover, if an IP is present in the test dataset but not in the data dataset, it considers connections only above the calculated average to be flagged as anomalous.

Test

As we can see multiple Devices increased by 10 times or more their traffic to the exterior. With the information from this rule and the rules internal_external_ip_conns, tcp_exfiltration and udp_exfiltration we can have more certainty that these devices are likely attempting to do a data exfiltration attack. Especially the Devices with IPs 192.168.106.144, 192.168.106.178, 192.168.106.38, 192.168.106.58, and 192.168.106.85 since these IPs are flagged in all 3 rules.



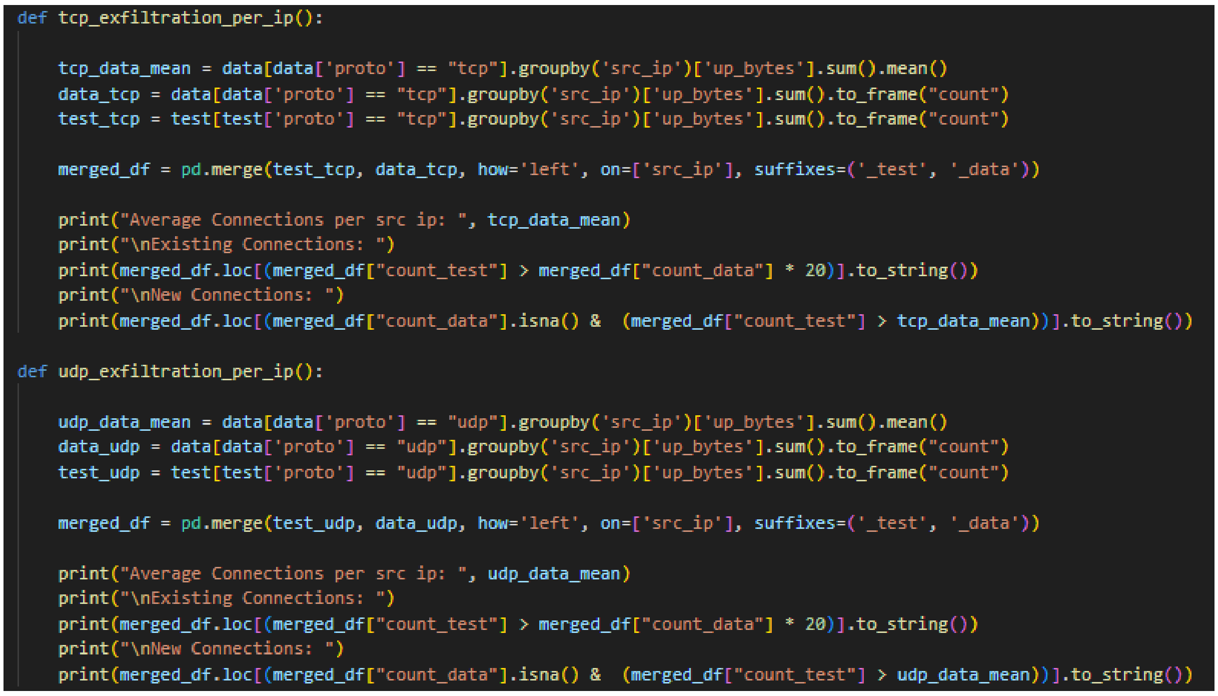
```
Average Connections per src ip:  3453.715736040609

Existing Connections:
                 count_test  count_data
src_ip
192.168.106.104        5999       548.0
192.168.106.142        2450       217.0
192.168.106.58         3886       279.0
192.168.106.85         4173        72.0

New Connections:
                 count_test  count_data
src_ip
192.168.106.144        4877         NaN
192.168.106.178        4993         NaN
192.168.106.38         4520         NaN
```

Figure 45: Internal to external IP connections Results

3.3.4 tcp_exfiltration_per_ip and udp_exfiltration_per_ip

The following Rules compute the number of upload bytes done by the HTTPS(TCP on port 443) and DNS(UDP on port 43) protocols.

The objective of this rule is to detect abnormal amounts of uploaded data for each Device which indicates data exfiltration.



```python
def tcp_exfiltration_per_ip():

    tcp_data_mean = data[data['proto'] == "tcp"].groupby('src_ip')['up_bytes'].sum().mean()
    data_tcp = data[data['proto'] == "tcp"].groupby('src_ip')['up_bytes'].sum().to_frame("count")
    test_tcp = test[test['proto'] == "tcp"].groupby('src_ip')['up_bytes'].sum().to_frame("count")

    merged_df = pd.merge(test_tcp, data_tcp, how='left', on=['src_ip'], suffixes=('_test', '_data'))

    print("Average Connections per src ip: ", tcp_data_mean)
    print("\nExisting Connections: ")
    print(merged_df.loc[(merged_df["count_test"] > merged_df["count_data"] * 20)].to_string())
    print("\nNew Connections: ")
    print(merged_df.loc[(merged_df["count_data"].isna() &  (merged_df["count_test"] > tcp_data_mean))].to_string())

def udp_exfiltration_per_ip():

    udp_data_mean = data[data['proto'] == "udp"].groupby('src_ip')['up_bytes'].sum().mean()
    data_udp = data[data['proto'] == "udp"].groupby('src_ip')['up_bytes'].sum().to_frame("count")
    test_udp = test[test['proto'] == "udp"].groupby('src_ip')['up_bytes'].sum().to_frame("count")

    merged_df = pd.merge(test_udp, data_udp, how='left', on=['src_ip'], suffixes=('_test', '_data'))

    print("Average Connections per src ip: ", udp_data_mean)
    print("\nExisting Connections: ")
    print(merged_df.loc[(merged_df["count_test"] > merged_df["count_data"] * 20)].to_string())
    print("\nNew Connections: ")
    print(merged_df.loc[(merged_df["count_data"].isna() &  (merged_df["count_test"] > udp_data_mean))].to_string())
```

Figure 46: Data Exfiltration IP

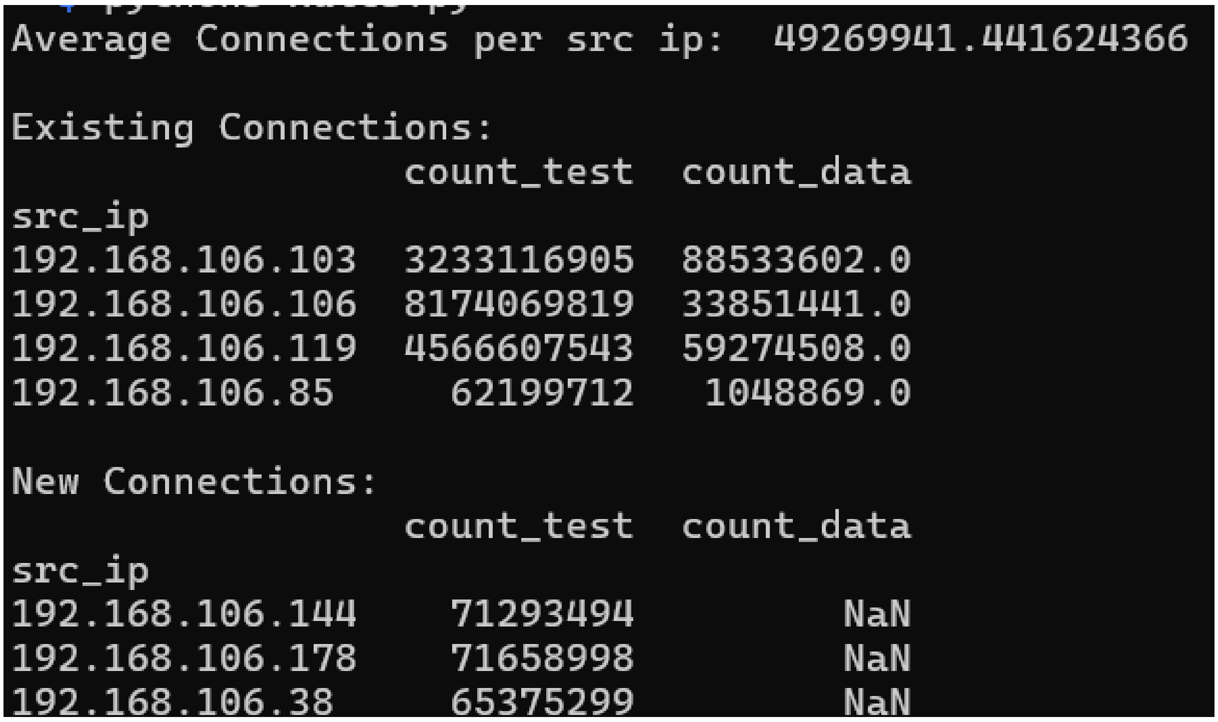
As shown above, the rules compute the average number of TCP/UDP uploaded bytes for each device, iT also calculates for the number of uploaded bytes for each IP in the data dataset and test dataset.

Then the functions compare the Devices from the test dataset that exceed the data daset values by 20 times.

Additionally, new Devices that exist only in the test dataset are flagged if their count surpasses the average number of uploads in the data dataset.

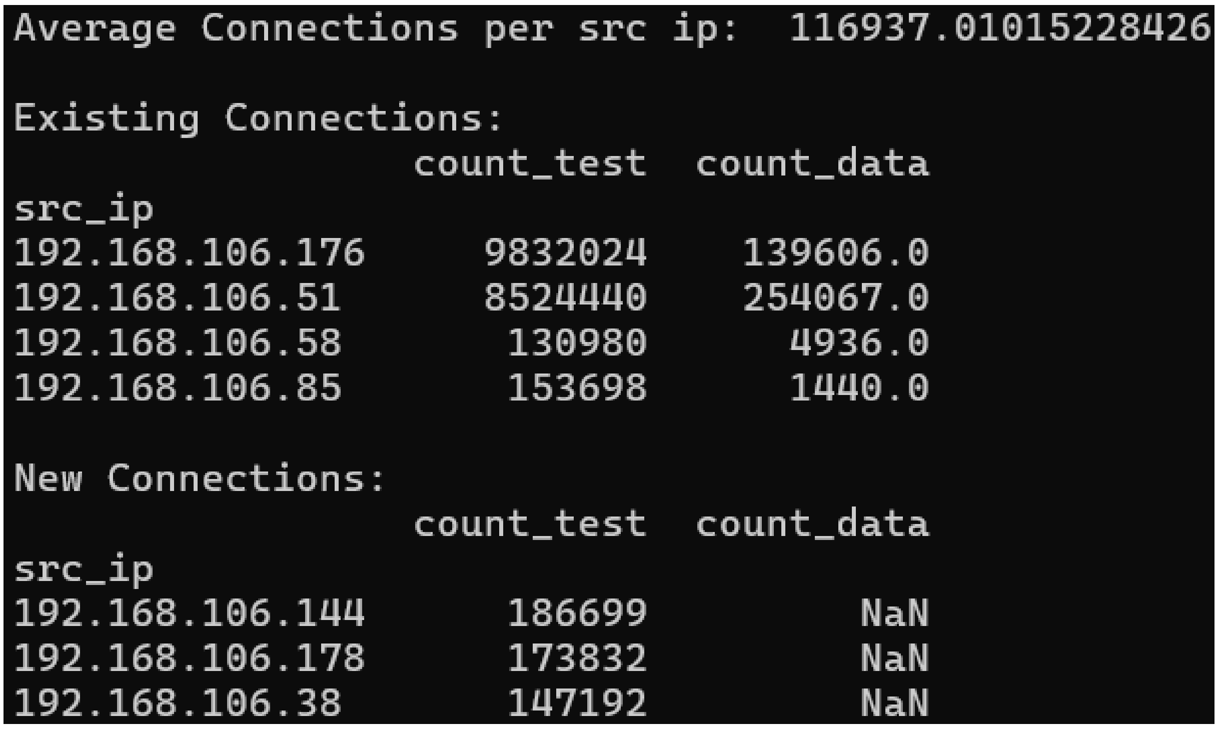
Test

As seen on the results below we can see multiple devices flagged, however, it is important to note that some devices had an increase in flow over 100 times which is the case of device 192.168.106.103, we can also see that Devices 192.168.106.144, 192.168.106.178, 192.168.106.38, 192.168.106.58 and 192.168.106.85 are once again being flagged in both these rules, further solidifying our believe that these rules are making some data exfiltration attack.



```
Average Connections per src ip:  49269941.441624366

Existing Connections:
                 count_test  count_data
src_ip
192.168.106.103  3233116905  88533602.0
192.168.106.106  8174069819  33851441.0
192.168.106.119  4566607543  59274508.0
192.168.106.85     62199712   1048869.0

New Connections:
                 count_test  count_data
src_ip
192.168.106.144    71293494         NaN
192.168.106.178    71658998         NaN
192.168.106.38     65375299         NaN
```

Figure 47: Tcp per IP results

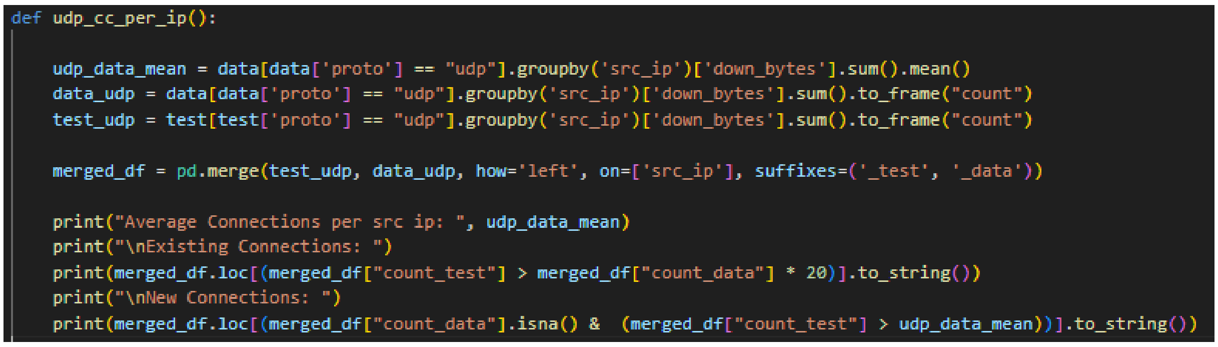


```
Average Connections per src ip:  116937.01015228426

Existing Connections:
                 count_test  count_data
src_ip
192.168.106.176     9832024    139606.0
192.168.106.51      8524440    254067.0
192.168.106.58       130980      4936.0
192.168.106.85       153698      1440.0

New Connections:
                 count_test  count_data
src_ip
192.168.106.144      186699         NaN
192.168.106.178      173832         NaN
192.168.106.38       147192         NaN
```

Figure 48: Udp per IP results

3.3.5 udp_cc_per_ip

The following rule has the objective of detecting unnatural download data for each device using the DNS protocol, which can indicate a C&C attack.



```
def udp_cc_per_ip():

    udp_data_mean = data[data['proto'] == "udp"].groupby('src_ip')['down_bytes'].sum().mean()
    data_udp = data[data['proto'] == "udp"].groupby('src_ip')['down_bytes'].sum().to_frame("count")
    test_udp = test[test['proto'] == "udp"].groupby('src_ip')['down_bytes'].sum().to_frame("count")

    merged_df = pd.merge(test_udp, data_udp, how='left', on=['src_ip'], suffixes=('_test', '_data'))

    print("Average Connections per src ip: ", udp_data_mean)
    print("\nExisting Connections: ")
    print(merged_df.loc[(merged_df["count_test"] > merged_df["count_data"] * 20)].to_string())
    print("\nNew Connections: ")
    print(merged_df.loc[(merged_df["count_data"].isna() &  (merged_df["count_test"] > udp_data_mean))].to_string())
```

Figure 49: UDP C&C IP

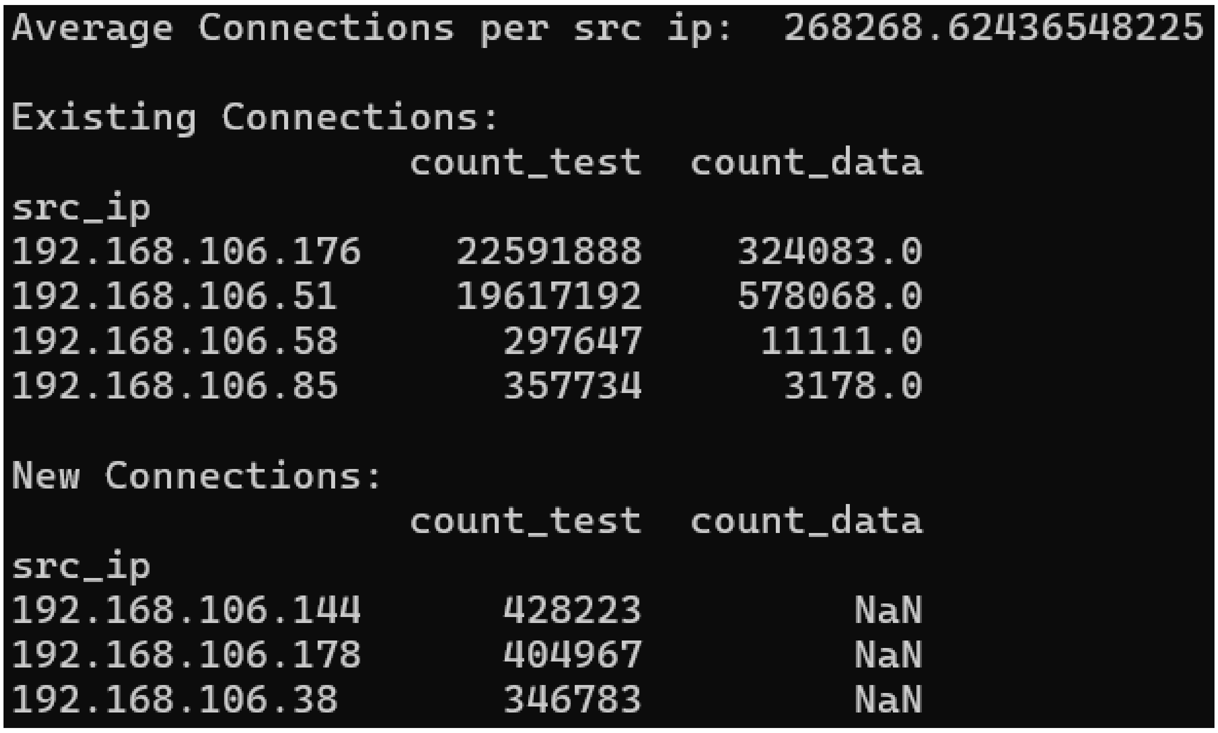
As shown above, the rules compute the average number of UDP download bytes for each device. They also calculate the downloaded bytes for each IP in both the data dataset and the test dataset.

Next, the functions compare the devices in the test dataset to those in the data dataset, flagging any that exceed the data values by 20 times.

Additionally, new devices that exist only in the test dataset are flagged if their upload count surpasses the average number of uploads in the data dataset.

Test

As seen in the results below some IPs like 192.168.106.176 and 192.168.106.51 had a increase of over 50 times and 192.168.106.85 had an increase of over 100 times. Using other results from previous rules such as udp_cc we see the IPs 192.168.106.176 and 192.168.106.51 in both, which can mean that they are making a C&C attack.



```
Average Connections per src ip:  268268.62436548225

Existing Connections:
                 count_test  count_data
src_ip
192.168.106.176    22591888    324083.0
192.168.106.51     19617192    578068.0
192.168.106.58       297647     11111.0
192.168.106.85       357734      3178.0

New Connections:
                 count_test  count_data
src_ip
192.168.106.144      428223         NaN
192.168.106.178      404967         NaN
192.168.106.38       346783         NaN
```

Figure 50: UDP C&C Results

3.4 Server

For the server, we implemented a rule similar to time_btw_requests rule, other rules created detected anomalies such as an abnormal number of connections, protocols other than ”TCP,” and activity on ports other than ”443.” We also attempted to detect the download and upload sizes.

However, as shown in the Server section, no devices were identified as anomalous.

We also attempted to detect attacks like DDoS, but no results were obtained.

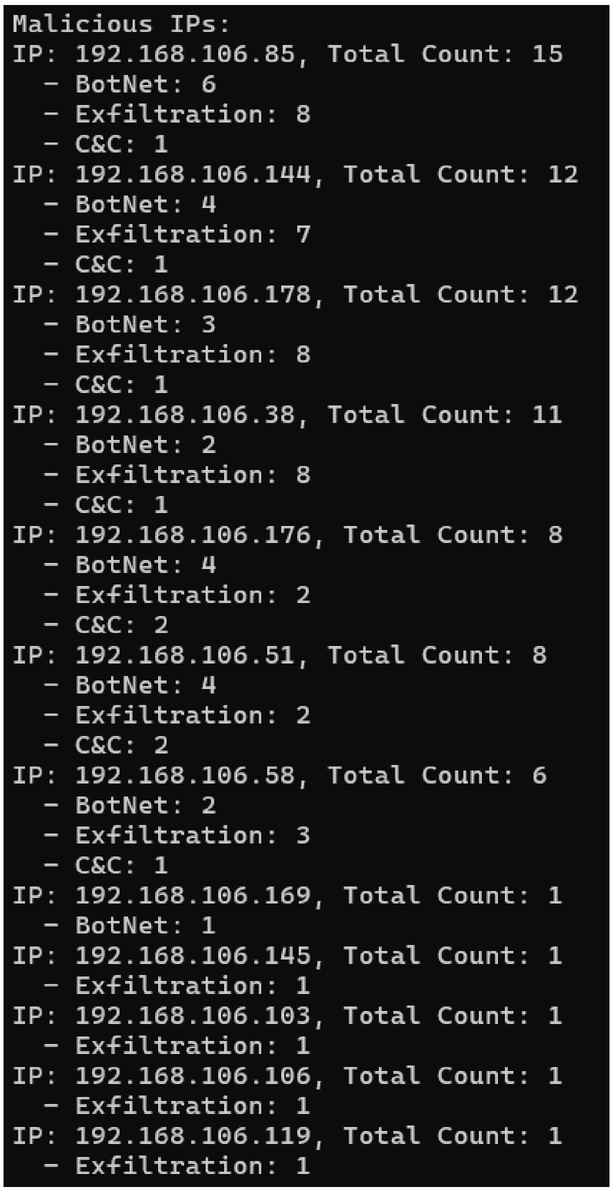
4 Device identification

The devices 192.168.106.85, 192.168.106.144, 192.168.106.178, and 192.168.106.38 have been flagged a remarkable 7 to 8 times due to suspicious exfiltration activities.

Regarding BotNet activities, multiple devices have been flagged as anomalous. Notably, 192.168.106.85, 192.168.106.144, 192.168.106.178, 192.168.106.176, 192.168.106.51, and 192.168.106.58 are probably associated with a BotNet. Additionally, even though 192.168.106.169 was flagged only once, its exceptionally high request frequency suggests potential involvement in the Bot-Net.

Furthermore, the devices 192.168.106.85, 192.168.106.144, 192.168.106.178, 192.168.106.38, 192.168.106.176, 192.168.106.51, and 192.168.106.58 have been flagged for possible involvement in a C&C attack.

In conclusion, while every flagged device demands attention, it's necessary to analyze each case individually. Some devices might demonstrate temporary unusual behavior rather than actively participating in malicious activities.

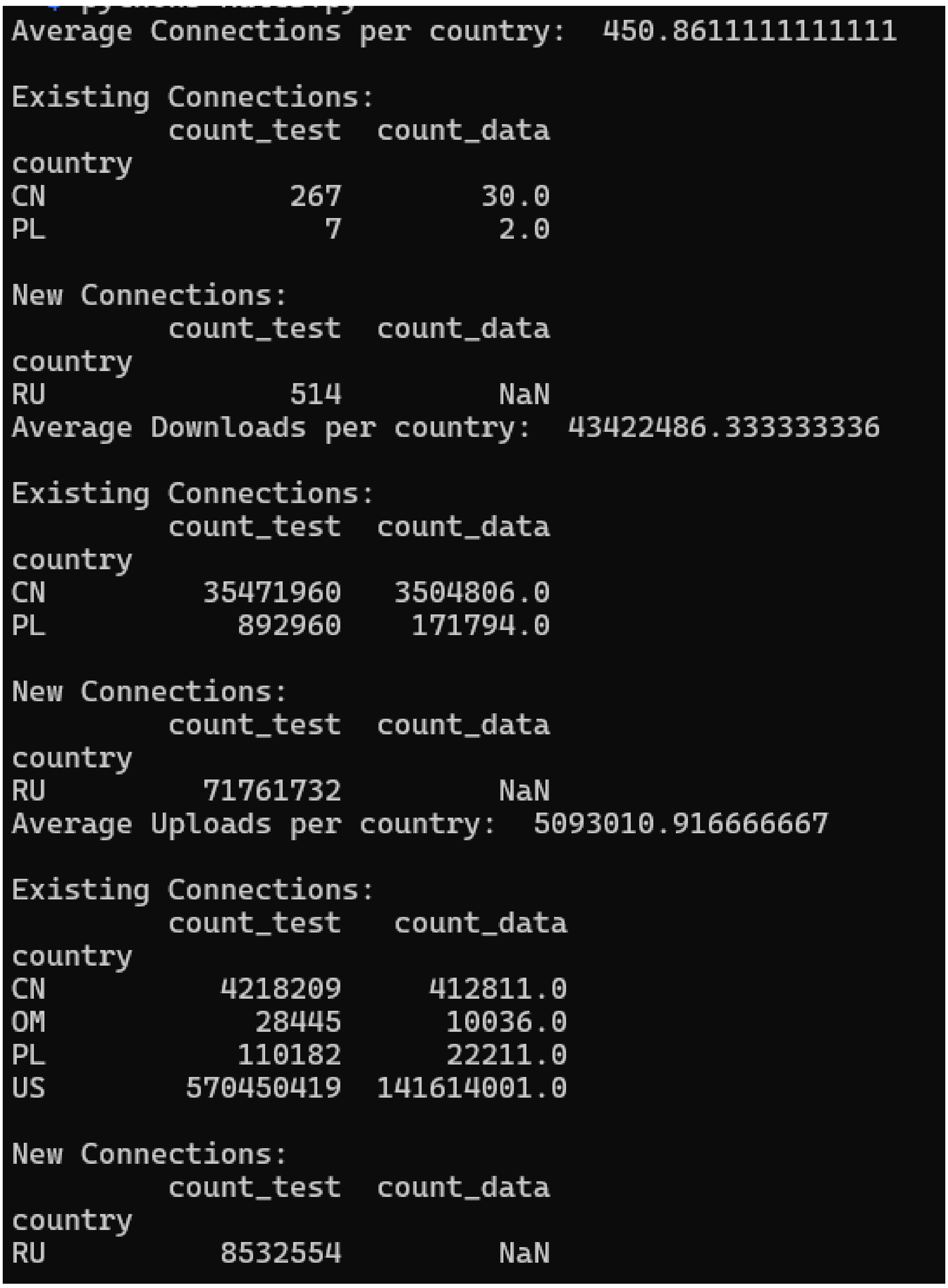


```
Malicious IPs:
IP: 192.168.106.85, Total Count: 15
  - BotNet: 6
  - Exfiltration: 8
  - C&C: 1
IP: 192.168.106.144, Total Count: 12
  - BotNet: 4
  - Exfiltration: 7
  - C&C: 1
IP: 192.168.106.178, Total Count: 12
  - BotNet: 3
  - Exfiltration: 8
  - C&C: 1
IP: 192.168.106.38, Total Count: 11
  - BotNet: 2
  - Exfiltration: 8
  - C&C: 1
IP: 192.168.106.176, Total Count: 8
  - BotNet: 4
  - Exfiltration: 2
  - C&C: 2
IP: 192.168.106.51, Total Count: 8
  - BotNet: 4
  - Exfiltration: 2
  - C&C: 2
IP: 192.168.106.58, Total Count: 6
  - BotNet: 2
  - Exfiltration: 3
  - C&C: 1
IP: 192.168.106.169, Total Count: 1
  - BotNet: 1
IP: 192.168.106.145, Total Count: 1
  - Exfiltration: 1
IP: 192.168.106.103, Total Count: 1
  - Exfiltration: 1
IP: 192.168.106.106, Total Count: 1
  - Exfiltration: 1
IP: 192.168.106.119, Total Count: 1
  - Exfiltration: 1
```

Figure 51: Malicious Devices

As for the countries, the enterprise should have a more attentive look into Devices coming from RU, CN, and PL. These countries have been flagged repeatedly, suggesting potential involvement in all three types of attacks.

Additionally, special attention should be directed towards devices from the US, where there has been a significant increase in downloads compared to the normal dataset. This spike can indicate an ongoing exfiltration attack.



```
Average Connections per country:  450.8611111111111

Existing Connections:
         count_test  count_data
country
CN              267        30.0
PL                7         2.0

New Connections:
         count_test  count_data
country
RU              514         NaN
Average Downloads per country:  43422486.333333336

Existing Connections:
         count_test  count_data
country
CN         35471960   3504806.0
PL           892960    171794.0

New Connections:
         count_test  count_data
country
RU         71761732         NaN
Average Uploads per country:  5093010.916666667

Existing Connections:
         count_test   count_data
country
CN          4218209     412811.0
OM            28445      10036.0
PL           110182      22211.0
US        570450419  141614001.0

New Connections:
         count_test  count_data
country
RU          8532554         NaN
```

Figure 52: Malicious Countrys

5 Conclusion

In conclusion, this project helped us understand how to better analyze traffic, define and test SIEM rules to detect anomalous network behaviors, and identify potentially compromised devices.

Additionally, we comprehended the importance of creating SIEM rules to identify and mitigate future attacks.

We believe the SIEM rules we developed are effective in accurately detecting malicious devices and identifying suspicious activities from specific countries.

However, to enhance their effectiveness, we recognize the need for a larger dataset with diverse use cases to better test and refine the parameters used.

Finally, It's essential to regularly review and update these rules to adapt to the evolving threat landscape and ensure the continuous protection of network resources.