SRC
Report Final Project

Universidade de Aveiro

Sebastian D. González, Mauro Filho

SRC
Report Final Project

Dept. de Eletrónica, Telecomunicações e Informática

Universidade de Aveiro

sebastian.duque@ua.pt(103690), mauro.filho@ua.pt(103411)

June 17, 2024

Contents

Introduction

In modern cybersecurity, monitoring network traffic is crucial for protecting organizational assets. This report focuses on defining and implementing Security Information and Event Management (SIEM) rules to detect anomalous behaviors and identify potentially compromised devices within a corporate network. Using historical traffic flow data, we aim to establish effective cybersecurity measures.

We analyze three Parquet files containing network traffic data:

- dataX.parquet: Historical data from internal devices, confirmed to be free of illicit behavior, used to establish normal network behavior.
- testX.parquet: Data from another day, potentially containing anomalies like botnet activities, data exfiltration, and remote command-and-control (C&C).
- serversX.parquet: Interactions between external clients and the corporation's public servers, used to identify anomalous external access patterns.

The project's objectives are:

- Analyze Non-Anomalous Behaviors: Identify internal servers and services, and quantify traffic exchanges within the network and with external servers using dataX.parquet.
- Define SIEM Rules: Create rules to detect botnet activities, data exfiltration, C&C activities, and anomalous external user behavior. Each rule will be justified based on observed patterns.
- Test SIEM Rules: Apply the rules to testX.parquet to identify devices showing anomalous behavior and evaluate the rules' effectiveness.

This report details our methods and findings, aiming to enhance the corporate network's cybersecurity defenses against potential threats.

Non-anomalous analysis

First lets start by analyzing the non-anomalous behaviour present in the file data1.parquet:

2.1 Internal Servers

In order to identify the internal servers from the dataset provided, we wrote the following script:

```python
### NON-ANOMALOUS BEHAVIOR ------------------------
# Identify internal servers/services
data_srcIPs = data['src_ip'].unique()
data_dstIPs = data['dst_ip'].unique()
internalNetwork = get_network(data_srcIPs[0])
internalServers = []
externalServers = []
for ip in data_dstIPs:
    isInternal = is_ip_in_network(ip, internalNetwork)
    if isInternal:
        internalServers.append(ip)
    else:
        externalServers.append(ip)
internalServers = remove_duplicates_set(internalServers)
externalServers = remove_duplicates_set(externalServers)
print("Internal servers: " + str(internalServers))
```

It takes the flows from the data file and, knowing that all source ips belong to the internal network, and verifies which ones of the destination ips belong to that same network. With that, we were able to discover 4 internal servers:

- '192.168.101.226', '192.168.101.224', '192.168.101.230', '192.168.101.239'

2.2 Traffic internal-internal

In order to extract some basic traffic analysis information from the dataset, we created the following function:

```python
def nonAnomalousAnalysisData(dataFrame):
    # Descriptive stats
    print("\nStats: ")
    print(dataFrame.describe())
    # Traffic volume over time
    traffic_time = dataFrame.groupby(pd.Grouper(key='
        timestamp')).size()
    plt.figure(figsize=(14, 7))
    plt.plot(traffic_time.index, traffic_time.values, label=
        'traffic')
    plt.xlabel('Time')
    plt.ylabel('Number of Packets')
    plt.title('Traffic Volume Over Time')
    plt.legend()
    plt.show()
    # Traffic distribution by destination IP
    traffic_distribution = dataFrame['dst_ip'].value_counts
        ()
    print("\nTraffic - destination count:\n",
        traffic_distribution)
    plt.figure(figsize=(12, 6))
    traffic_distribution.head(15).plot(kind='bar')
    plt.xlabel('Destination IP')
    plt.ylabel('Frequency')
    plt.title('Traffic - destination count')
    plt.show()
    # Distribution by country
    traffic_countries = dataFrame['country'].value_counts()
    print("\nTraffic - destination country:\n",
        traffic_countries)
    # Distribution by protocols
    traffic_protocols = dataFrame["proto"].value_counts()
    print("\nTraffic - protocols:\n", traffic_protocols)
    # Distribution by port
    traffic_port = dataFrame["port"].value_counts()
    print("\nTraffic - port:\n", traffic_port)
```

Which will be used to extract the same information for traffic internal-internal, internal-external, and external-public servers. This code provide us with the following information:

- General stats
- Number of flows though time
- Traffic destination distribution
- Traffic distribution by protocol
- Traffic distribution by port

In this section we shall explore those findings for the traffic internal-internal.

Starting with the general stats:

```
Stats:
            timestamp          port      up_bytes    down_bytes
count  2.84995e+05  284995.0000  284995.0000  2.849950e+05
mean   4.75702e+06     286.1857    6892.1437  6.332180e+04
std    1.29164e+06     191.2249    7322.9693  1.263935e+05
min    1.43291e+06      53.0000      72.0000  7.500000e+01
25%    3.80541e+06      53.0000     209.0000  4.860000e+02
50%    4.70365e+06     443.0000    6057.0000  4.841900e+04
75%    5.65783e+06     443.0000   11079.0000  9.344550e+04
max    8.48567e+06     443.0000  101701.0000  8.421091e+06
```

We can observe at least 2 ports being used internally: 53 and 443, and also some interesting data regarding the bytes uploaded and downloaded, like their maximum, total and average values.

Moving on to the number of flows through time:

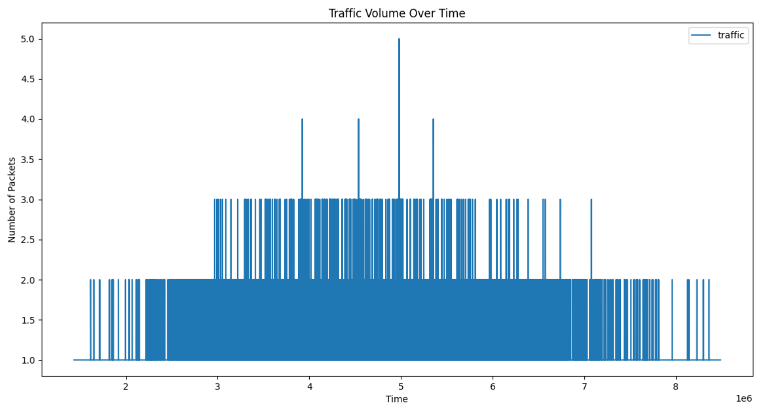

Figure 2.1: Number of flows through time, internal-internal

We can observe a fairly constant count of at least 2 flows at any given time of the day, with a busier period of communications between the timestamps 3 and 7, and a maximum of 5 flows at the same time around the middle of the day.

When it comes to destinations, we are able to verify the previous affirmation about the internal servers, by seeing that those are the only destination IPs in the internal traffic:

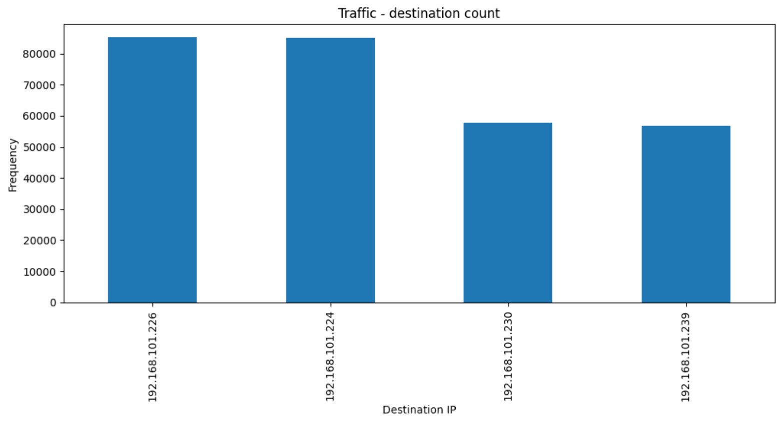

Figure 2.2: Distribution of destinations, internal-internal

When observing the distribution by ports, we are once again able to confirm a previous piece of data shown by general stats:

```
Traffic - port:
port    count
443    170402
53     114593
```

And we see the same numbers when observing the distribution of flows by protocol:

```
Traffic - protocols:
proto   count
tcp    170402
udp    114593
```

Which means we can assume the flows on port 443 were accessing an HTTPS server and flows on port 53 were accessing a DNS server.

2.3 Traffic internal-external

Utilizing the same function mentioned before, we are able to observe some information on internal-external traffic:

```
Stats:
           timestamp      port        up_bytes    down_bytes
count   6.745520e+05  674552.0   674552.000000  6.745520e+05
mean    4.768591e+06     443.0    11384.964585  1.051845e+05
std     1.288769e+06       0.0     6277.647516  1.479753e+05
min     1.433011e+06     443.0      593.000000  3.815000e+03
25%     3.817127e+06     443.0     7058.000000  5.716400e+04
```

```
50%     4.718850e+06      443.0     9973.000000   8.317700e+04
75%     5.661680e+06      443.0    14081.000000   1.211170e+05
max     8.487545e+06      443.0   131994.000000   7.862766e+06
```

When it comes to flows through time, we can observe that, similarly to the internal traffic, there is a constant minimum of at least 2 flows throughout the day, with the peak of flows being between 4 and 6 in the timestamp scale.

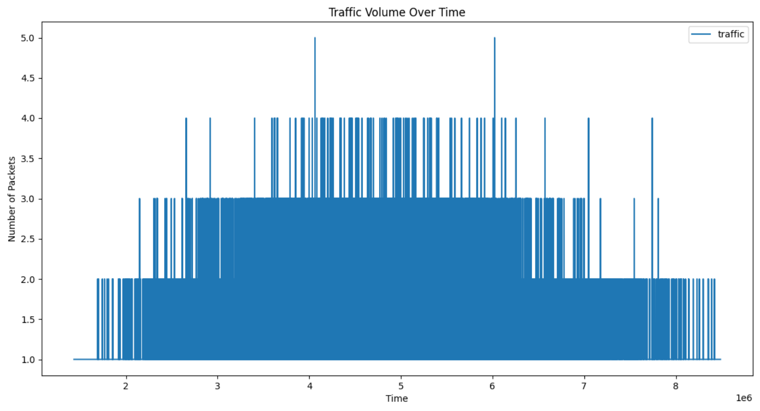

Figure 2.3: Number of flows through time, internal-external

Differently from the internal traffic, which only had 4 internal servers being accessed, now on the external traffic we got many more. In the following graph we show the top 15 destination addresses with the most total flows:

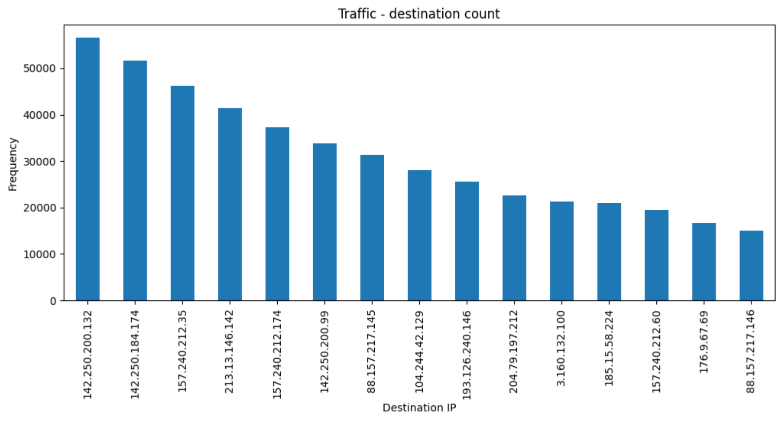

Figure 2.4: Distribution of destinations, internal-external

When observing the distribution by ports, we can observe this:

```
Traffic - port:
port     count
443     674552
```

And we see the same numbers when observing the distribution of flows by protocol:

```
Traffic - protocols:
proto    count
tcp     674552
```

Which means we can assume the flows from internal to external are all accessing HTTPS servers.

When looking at the external traffic, we can also take a look at the countries which are receiving theses communications from internals addresses the most. The following list describes the 10 most popular traffic destinations in terms of number of flows:

```
Traffic - destination country:
country  count
US      337317
PT      258111
NA       19604
NL       19494
DE       17725
GB        8221
ES        5354
BR        4209
IE         681
IN         539
```

As we can see, the United States is in first place, followed by Portugal and Namibia, respectively.

When observing this external traffic we also added an extra piece of information regarding the upload of data from internal devices to the outside, so we added this function to our script, which will tell us the top 10 internal "uploaders", and where they are uploading to:

```
def topUploadersAnalysis(normalDataFrame):
    normal_up_bytes = normalDataFrame.groupby('src_ip')['
        up_bytes'].mean()
    top_uploaders = normal_up_bytes.sort_values(ascending=
        False).head(10)
    org_uploads = {}
    country_uploads = {}
    for ip in top_uploaders.index:
        avg_bytes = top_uploaders[ip]
        print(f"IP: {ip}, Average Upload Bytes: {avg_bytes}"
            )
```

```python
    print("\n")
    for ip in top_uploaders.index:
        ip_data = normalDataFrame[normalDataFrame['src_ip'] == ip]
        dst_ip_bytes = ip_data.groupby('dst_ip')['up_bytes'].sum().sort_values(ascending=False)
        orgs = dst_ip_bytes.index.map(lambda x: gi2.org_by_addr(x))
        countries = dst_ip_bytes.index.map(lambda x: gi.country_code_by_addr(x))
        for org, bytes_received in zip(orgs, dst_ip_bytes):
            if org in org_uploads:
                org_uploads[org] += bytes_received
            else:
                org_uploads[org] = bytes_received
        for country, bytes_received in zip(countries, dst_ip_bytes):
            if country in country_uploads:
                country_uploads[country] += bytes_received
            else:
                country_uploads[country] = bytes_received
    top_orgs = sorted(org_uploads.items(), key=lambda x: x[1], reverse=True)[:10]
    top_countries = sorted(country_uploads.items(), key=lambda x: x[1], reverse=True)[:10]
    print("Top 10 Organizations Receiving Uploads:")
    for org, bytes_received in top_orgs:
        print(f"Organization: {org}, Total Bytes: {bytes_received}")
    print("\nTop 10 Countries Receiving Uploads:")
    for country, bytes_received in top_countries:
        print(f"Country: {country}, Total Bytes: {bytes_received}")
```

Which gives us the results:

```
IP: 192.168.101.199, Average Upload Bytes: 11902.58747993
IP: 192.168.101.185, Average Upload Bytes: 11830.03179753
IP: 192.168.101.104, Average Upload Bytes: 11718.32978723
IP: 192.168.101.36,  Average Upload Bytes: 11695.66666666
IP: 192.168.101.130, Average Upload Bytes: 11658.97267759
IP: 192.168.101.13,  Average Upload Bytes: 11643.95449620
IP: 192.168.101.39,  Average Upload Bytes: 11630.23409196
IP: 192.168.101.64,  Average Upload Bytes: 11622.20120967
IP: 192.168.101.124, Average Upload Bytes: 11618.89049919
IP: 192.168.101.87,  Average Upload Bytes: 11616.82509328
```

As we can see, their average uploaded bytes are not that distant from each other.

Now when talking about where this data is being uploaded to, we can take a look at the rest of the information given by the script:

```
Top 10 Organizations Receiving Uploads:
Org: AS15169 GOOGLE,              Total Bytes: 39736018
Org: AS32934 FACEBOOK,            Total Bytes: 29088964
Org: AS2860 Nos Comunicacoes, S.A., Total Bytes: 20893558
Org: AS3243 Servicos De Comunic..., Total Bytes: 11061452
Org: AS8068 MICROSOFT-CORP-MSN-..., Total Bytes: 10044381
Org: AS8075 MICROSOFT-CORP-MSN-..., Total Bytes: 8632603
Org: AS13414 TWITTER,             Total Bytes: 7583823
Org: AS15525 Servicos De Comuni..., Total Bytes: 7190172
Org: AS13335 CLOUDFLARENET,       Total Bytes: 6023062

Top 10 Countries Receiving Uploads:
Country: US, Total Bytes: 94049328
Country: PT, Total Bytes: 73032552
Country: DE, Total Bytes: 5457407
Country: NA, Total Bytes: 4956780
Country: NL, Total Bytes: 4224997
Country: ES, Total Bytes: 1988124
Country: GB, Total Bytes: 1797536
Country: BR, Total Bytes: 1451399
Country: AU, Total Bytes: 265013
Country: SG, Total Bytes: 114505
```

2.4 Traffic external-public servers

Now looking at the data from the servers file we observe:

```
Stats:
          timestamp      port       up_bytes    down_bytes
count  7.140340e+05  714034.0  714034.000000  7.140340e+05
mean   4.723020e+06     443.0   11317.586926  9.618652e+04
std    1.291634e+06       0.0    6165.639099  5.675076e+04
min    1.340071e+06     443.0     841.000000  6.938000e+03
25%    3.774257e+06     443.0    7047.000000  5.708800e+04
50%    4.624560e+06     443.0    9931.000000  8.275800e+04
75%    5.584250e+06     443.0   14024.000000  1.202058e+05
max    8.489793e+06     443.0   98929.000000  1.014845e+06
```

And by looking at this information:

```
Traffic - destination country:
country  count
PT      714034

Traffic - protocols:
proto   count
tcp    714034
```

```

Traffic - port:
port     count
443     714034
```

We can infer that the public server of the company is located in portugal is processing only HTTPS requests Now by looking at the timeline of flows through the timestamps data, we gather that most accesses happen around the middle of the day, which is being confirmed to be a trend between all the graphics analyzed so far

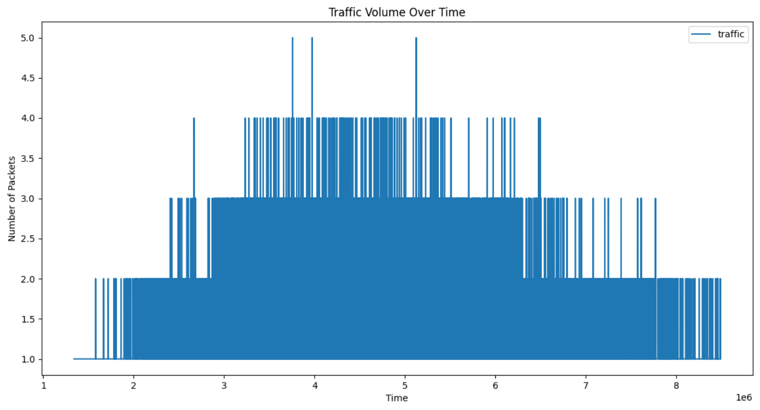

Figure 2.5: Flows through time, external-server

Then when looking at the distribution of traffic by the destination, we found that there are two ip addresses handling these requests:

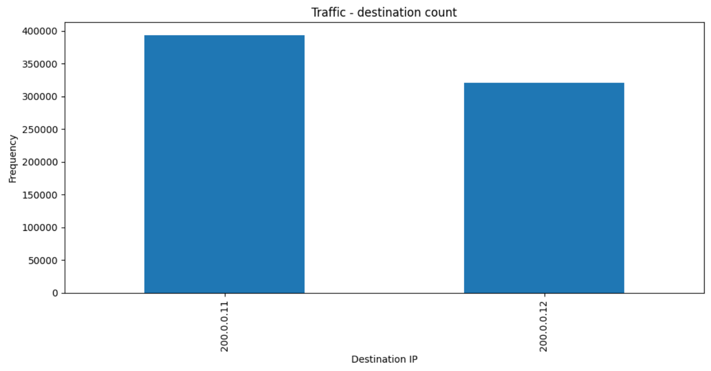

Figure 2.6: Distribution of destinations, external-server

SIEM rules for identifying anomalous behaviour

3.1 Internal BotNet activities

Botnets consist of compromised devices controlled by a malicious actor, often revealing their presence through unusual network traffic patterns.

To start, it's essential to understand what normal network behavior looks like in the environment. This involves establishing a baseline for typical traffic volume, destinations, protocols, and behavior patterns. This has already been done through our analysis of the normal data provided in the dataset, however further analysis may be needed in order to draw better conclusions.

BotNet activity can be identified by taking a closer look at both the activity between two or more internal devices, and between an internal device and an external one. Internal devices that are compromised will communicate with each other in an unusual way, and will also periodically communicate with suspicious external.

Therefore, we could establish two SIEM rules:

- Trigger an alert when internal devices that dont usually communicate, start communicating regularly
- Trigger an alert when an internal device starts communicating regularly with the same external devices

3.2 Data exfiltration

Data exfiltration involves unauthorized transfer of data from a system, often masked through legitimate protocols like DNS and HTTPS. Identifying such activities requires a deep understanding of normal protocol usage patterns within the environment. This begins with establishing a baseline for typical DNS and HTTPS traffic, including volume, destinations, and frequency. Our prior analysis of the normal dataset has laid the groundwork for this, though additional analysis may be beneficial for more nuanced detection.

Data exfiltration can be detected by closely examining the patterns of DNS and HTTPS traffic. Malicious actors often use DNS tunneling to hide exfiltration within seemingly normal DNS queries, or they leverage HTTPS to encrypt and conceal the data being sent to external servers. Identifying these anomalies involves monitoring for unusual query patterns, data volumes, and the nature of the endpoints involved.

Therefore, we could establish two SIEM rules:

- Trigger an alert when an internal device generates an unusually high volume of DNS queries, especially to external domains not commonly contacted.
- Trigger an alert when an internal device starts transmitting large amounts of data over HTTPS to external destinations not frequently accessed.

3.3 C&C activities using DNS

In a Command and Control (C&C) attack, compromised systems typically establish communication with a command server to receive instructions or transmit data. An unusual surge in network traffic from specific IP addresses to these servers may indicate that these systems have been compromised and are part of a botnet used to execute the C&C attack. This unusual traffic pattern can be a key indicator of a potential security breach, allowing network administrators to identify and respond to the threat more effectively.

In our Project we analysed C&C attacks using DNS. In this scenario attackers can, for example, use DNS to disguise their malicious communications by mimicking legitimate DNS queries. Therefore, we could establish this SIEM rule:

- Trigger an alert for potential malicious C&C activity. This rule identifies malicious DNS traffic patterns by monitoring DNS queries for unusual patterns and anomalies.

3.4 External users using the corporate public services in an anomalous way

Anomalous external communications is a serious threat to an organization security. Its essential to monitor all external connections for protecting sensitive information, preventing data breaches etc. In our project we identified all new IP addresses communicating externally and proceeded to a deep analysis to investigate the volume of data these IPs are uploading and downloading. We also evaluated if the uploading and downloading are occurring outside of normal business hours [1] since attackers often perform malicious activities outside of regular business hours to evade detection.

- Trigger an alert for potential external malicious activity. This rule identifies malicious external IP addresses monitoring data exchange(uploading and downloading).

Identifying devices with anomalous behaviour

4.1 Internal BotNet activities

In order to identify devices that might be partaking in suspicious BotNet activities we created this script:

```python
def internalBotNetAnalysis(dataFrameData, dataFrameTest):
    # Gets all dst ips for each src ip in normal data
    normal_src_to_dst_dict = {}
    for src_ip in dataFrameData['src_ip'].unique():
        src_data = dataFrameData[dataFrameData['src_ip'] ==
            src_ip]
        top_dst_ips = src_data['dst_ip'].value_counts().
            index.tolist()
        normal_src_to_dst_dict[src_ip] = top_dst_ips
    # Gets all dst ips for each src ip in test data
    test_src_to_dst_dict = {}
    for src_ip in dataFrameTest['src_ip'].unique():
        src_data = dataFrameTest[dataFrameTest['src_ip'] ==
            src_ip]
        top_dst_ips = src_data['dst_ip'].value_counts().
            index.tolist()
        test_src_to_dst_dict[src_ip] = top_dst_ips
    # Gets, for each new src ip or src ip that communicates
        with new dst ips, the dst ips that dont match the
        normal data dst ips
    suspicious_ips = {}
    for src_ip in test_src_to_dst_dict:
        if src_ip not in normal_src_to_dst_dict:
            suspicious_ips[src_ip] = test_src_to_dst_dict[
                src_ip]
            print(f"Source IP {src_ip} not found in normal
                data.")
        else:
            normal_dst_ips = set(normal_src_to_dst_dict[
                src_ip])
            test_dst_ips = set(test_src_to_dst_dict[src_ip])
            if normal_dst_ips != test_dst_ips:
                different_ips = test_dst_ips -
                    normal_dst_ips
                if different_ips:
                    print(f"Source IP {src_ip} has
                        mismatched destination IPs: {
                        different_ips}")
                suspicious_ips[src_ip] = different_ips
    print("\nSUSPICIOIUS IPS:")
    print(suspicious_ips)
    # Filter suspicious IPs based on mutual communication
    filtered_suspicious_ips = {}
    for src_ip, dst_ips in suspicious_ips.items():
        for dst_ip in dst_ips:
            if dst_ip in suspicious_ips and src_ip in
                suspicious_ips[dst_ip]:
                if src_ip not in filtered_suspicious_ips:
                    filtered_suspicious_ips[src_ip] = []
                filtered_suspicious_ips[src_ip].append(
                    dst_ip)
    suspicious_ips = filtered_suspicious_ips
    print("\nSUSPICIOUS IPS THAT COMMUNICATE WITH OTHER
        SUSPICIOUS INTERNAL IPS:")
    print(suspicious_ips)
```

This function is designed to analyze network traffic data to identify potentially suspicious IP addresses based on their communication patterns. It takes in two datasets: one containing normal traffic data (dataFrameData) and another containing test traffic data (dataFrameTest).

Initially, the function builds a dictionary for both the normal and test datasets. Each dictionary maps source IP addresses to a list of their most frequently contacted destination IP addresses. For each source IP in the normal data, it records the destination IPs it communicates with most often. Similarly, it does the same for the test data.

Next, the function compares the two dictionaries to identify source IP addresses in the test data that either do not appear in the normal data or communicate with destination IPs that are not present in the normal data. For any source IPs with such discrepancies, it flags them as suspicious and records the destination IPs that caused the mismatch.

Finally, the function filters these suspicious IPs further by checking for mutual communication among them. If a suspicious source IP communicates with another suspicious source IP, both IPs are noted as having a suspicious mutual communication pattern. This step aims to refine the list of potentially malicious IPs by highlighting those that exhibit coordinated communication behavior, which is often indicative of botnet activity.

By running this code on the two datasets available, these are the results we get:

```
SUSPICIOIUS IPS:
{
-'192.168.101.20': ['192.168.101.226', '192.168.101.239',
    '192.168.101.224', '192.168.101.230'],
-'192.168.101.33': {'192.168.101.226', '192.168.101.230'},
    '192.168.101.57': ['192.168.101.224', '192.168.101.226',
    '192.168.101.239', '192.168.101.230'],
-'192.168.101.169': {'192.168.101.118', '192.168.101.131'},
-'192.168.101.209': ['192.168.101.224', '192.168.101.226',
    '192.168.101.230', '192.168.101.239'],
-'192.168.101.118': {'192.168.101.169', '192.168.101.131'},
-'192.168.101.131': {'192.168.101.118', '192.168.101.169'}
}

SUSPICIOUS IPS THAT COMMUNICATE WITH OTHER SUSPICIOUS
    INTERNAL IPS:
{
-'192.168.101.169': ['192.168.101.118', '192.168.101.131'],
-'192.168.101.118': ['192.168.101.169', '192.168.101.131'],
-'192.168.101.131': ['192.168.101.118', '192.168.101.169']
}
```

The first list of suspicious IPs are IPs that communicate with different destinations when compared to the normal data, and then the second list filters them by identifying the ones that communicate with other suspicious IPs, which shows a behaviour of coordinated communication between possible members of the BotNet.

In order to confirm the suspicious behaviour, we also generated graphics of the messages exchanged between the suspicious nodes:

```python
for sender_ip, receiver_ips in suspicious_ips.items():
    plt.figure(figsize=(10, 5))
    plt.title(f"Communication Timeline - Suspicious Sender
        IP: {sender_ip}")
    plt.xlabel("Time")
    plt.ylabel("Number of Messages")
    for receiver_ip in receiver_ips:
        sender_data = dataFrameTest[dataFrameTest['src_ip']
            == sender_ip]
        receiver_data = sender_data[sender_data['dst_ip'] ==
             receiver_ip]
        plt.plot(receiver_data['timestamp'], range(len(
            receiver_data)), label=f"Receiver IP: {
            receiver_ip}")
    plt.legend()
    plt.show()
```

Which gives us the following:

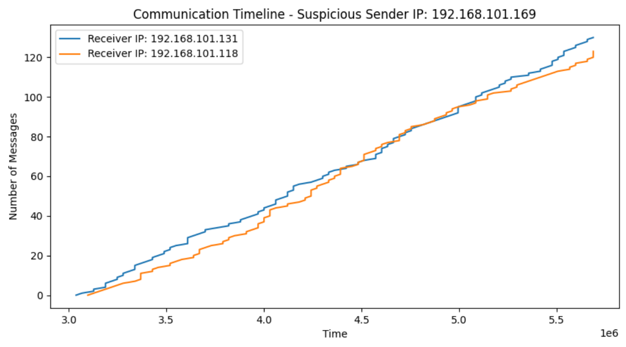

Figure 4.1: Timeline botnet suspicious sender 1

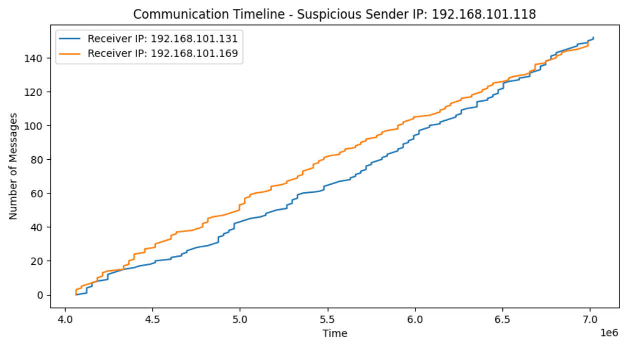

Figure 4.2: Timeline botnet suspicious sender 2

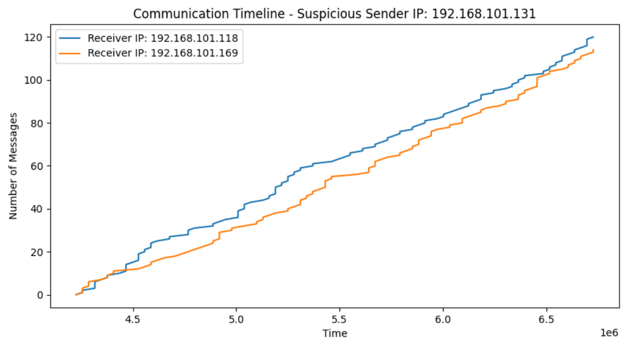

Figure 4.3: Timeline botnet suspicious sender 3

And like this we can confirm that they have been communicating frequently throughout the day and they are probably working together as a part of the BotNet.

4.2 Data exfiltration

In order to analyze the traffic data in search of devices that migh be partaking in data exfiltration, we developed the function:

```python
def dataExfiltrationAnalysis(normalDataFrame, testDataFrame):
    normal_up_bytes = normalDataFrame.groupby('src_ip')['up_bytes'].mean()
    test_up_bytes = testDataFrame.groupby('src_ip')['up_bytes'].mean()
    increases = []
    for ip in test_up_bytes.index:
        normal_bytes = normal_up_bytes.get(ip, 0)
        test_bytes = test_up_bytes[ip]
        increase_percentage = ((test_bytes - normal_bytes) / normal_bytes * 100) if normal_bytes > 0 else float('inf')
        if increase_percentage >= 100:
            increases.append((ip, normal_bytes, test_bytes, increase_percentage))
    increases.sort(key=lambda x: x[3], reverse=True)
    IpsByOrgBySrcIp = {}
    for ip, normal_bytes, test_bytes, increase_percentage in increases:
        print(f"IP: {ip}, Normal: {normal_bytes}, Test: {test_bytes}, Increase: {increase_percentage:.2f}%")
        ip_data = testDataFrame[testDataFrame['src_ip'] == ip]
        dst_ip_bytes = ip_data.groupby('dst_ip')['up_bytes'].sum().sort_values(ascending=False)
        # Orgs receiving the most bits
        orgs = dst_ip_bytes.index.map(lambda x: gi2.org_by_addr(x))
        org_bytes = dst_ip_bytes.groupby(orgs).sum().sort_values(ascending=False)
        # IPs by org
        org_dst_map = {}
        for org, dst_ip in zip(orgs, dst_ip_bytes.index):
            if org not in org_dst_map:
                org_dst_map[org] = {}
            org_dst_map[org][dst_ip] = ip_data[ip_data['dst_ip'] == dst_ip]['up_bytes'].sum()
        finalOrgMap = {}
        for org, bytes_received in org_bytes.head(3).items()
            :
            if bytes_received > 0:
                finalOrgMap[org] = org_dst_map[org]
        IpsByOrgBySrcIp[ip] = finalOrgMap
    print("\n")
    for srcIp, orgDic in IpsByOrgBySrcIp.items():
        total_uploaded_bytes = testDataFrame[testDataFrame['
            src_ip'] == srcIp]['up_bytes'].sum()
        if total_uploaded_bytes > 500 * 1000000:  # Check if
             total uploaded bytes exceed 500 MB
            print("Src IP: " + srcIp + " Total upload: " +
                str(total_uploaded_bytes))
            for org, ipDict in orgDic.items():
                print("    Org: " + org)
                for dstIp, dstBytes in ipDict.items():
                    print("        Dst IP: " + dstIp + ",
                        Total bytes: " + str(dstBytes))
```

This function, aims to detect potential data exfiltration activities within network traffic data. It begins by calculating the average upload bytes per source IP for both the normal and test datasets. It then identifies significant increases in upload bytes in the test data compared to the normal data, considering a threshold of at least a 100% increase in average upload size.

For each IP address with a substantial increase, it prints out details such as the IP address, the normal and test upload bytes, and the percentage increase. Additionally, it analyzes the destinations of the increased traffic, categorizing them by organization and presenting the top three organizations receiving the most data. It also lists the specific destination IP addresses and the amount of data each received.

Finally, it checks if the total uploaded bytes from each source IP in the test data exceed 500 MB. If so, it prints out the source IP along with the total upload amount, as well as the organizations and destination IP addresses involved in the upload, providing insights into potentially suspicious data exfiltration activities.

By running the function on the given datasets, we get the following output:

```
IP: 192.168.101.20, Normal: 0, Test: 11478.010227272727,
    Increase: inf%
IP: 192.168.101.209, Normal: 0, Test: 11394.192419121198,
    Increase: inf%
IP: 192.168.101.57, Normal: 0, Test: 11301.643436754177,
    Increase: inf%
IP: 192.168.101.155, Normal: 11414.9718781386, Test:
    1508304.161520962, Increase: 13113.38%
IP: 192.168.101.147, Normal: 11331.533429302623, Test:
    1003827.0790378007, Increase: 8758.70%
IP: 192.168.101.174, Normal: 11318.809300508012, Test:
    40048.731492225575, Increase: 253.82%
```

```
IP: 192.168.101.30, Normal: 11480.479714030384, Test:
    32121.759406282363, Increase: 179.79%
IP: 192.168.101.152, Normal: 11520.528828374518, Test:
    28984.776916451334, Increase: 151.59%


Src IP: 192.168.101.155 Total upload: 4641051905
    Org: AS15169 GOOGLE
        Dst IP: 142.250.184.243, Total bytes: 4606823380
        Dst IP: 142.250.200.132, Total bytes: 3277698
        Dst IP: 142.250.184.174, Total bytes: 1866208
        Dst IP: 142.250.200.99, Total bytes: 1790964
        Dst IP: 208.68.108.51, Total bytes: 9330
    Org: AS2860 Nos Comunicacoes, S.A.
        Dst IP: 88.157.217.145, Total bytes: 2712838
        Dst IP: 193.126.240.146, Total bytes: 1565333
        Dst IP: 88.157.217.146, Total bytes: 580076
        Dst IP: 88.157.217.148, Total bytes: 194651
    Org: AS32934 FACEBOOK
        Dst IP: 157.240.212.35, Total bytes: 1954028
        Dst IP: 157.240.212.174, Total bytes: 1910998
        Dst IP: 157.240.212.60, Total bytes: 1123814
        Dst IP: 179.60.193.45, Total bytes: 53253
Src IP: 192.168.101.147 Total upload: 2336909440
    Org: AS15169 GOOGLE
        Dst IP: 142.250.184.226, Total bytes: 2310094454
        Dst IP: 142.250.184.174, Total bytes: 2171779
        Dst IP: 142.250.200.132, Total bytes: 2106129
        Dst IP: 142.250.200.99, Total bytes: 1359994
        Dst IP: 199.36.155.227, Total bytes: 73715
        Dst IP: 172.110.38.227, Total bytes: 14091
    Org: AS32934 FACEBOOK
        Dst IP: 157.240.212.35, Total bytes: 2116174
        Dst IP: 157.240.212.174, Total bytes: 1662859
        Dst IP: 157.240.212.60, Total bytes: 804910
    Org: AS2860 Nos Comunicacoes, S.A.
        Dst IP: 88.157.217.145, Total bytes: 1577488
        Dst IP: 193.126.240.146, Total bytes: 1303902
        Dst IP: 88.157.217.146, Total bytes: 362244
        Dst IP: 88.157.217.148, Total bytes: 77855
```

We can observe on this output that 8 devices have experienced over 100% increases in their upload bytes, however, only 2 have uploaded more than 500mb to the exterior world, which tells us that those are the most suspicious ones.

The device with source IP ending in 155 uploaded over 4gb of data to the outside, in an outrageous increase of 13113.38% in average upload size when compared to the normal data. The device with source IP ending in 147 uploaded over 2gb of data to the outside, in an equally shocking increase of 8758.70% in average upload size when compared to the normal data.

When looking at the receivers of said data, we can observe that most of the data uploaded from the internal devices have reached the IPs: 142.250.184.226 and 142.250.184.243, both belonging to the organization AS15169 GOOGLE.

4.3 C&C activities using DNS

From our analysis we notice that there were a pair of IP addresses that clearly were under an attack. This conclusion was drawn from an unusual increase in the number of flows targeting these server IPs(226 and 224) as we can see in figure 4.4.

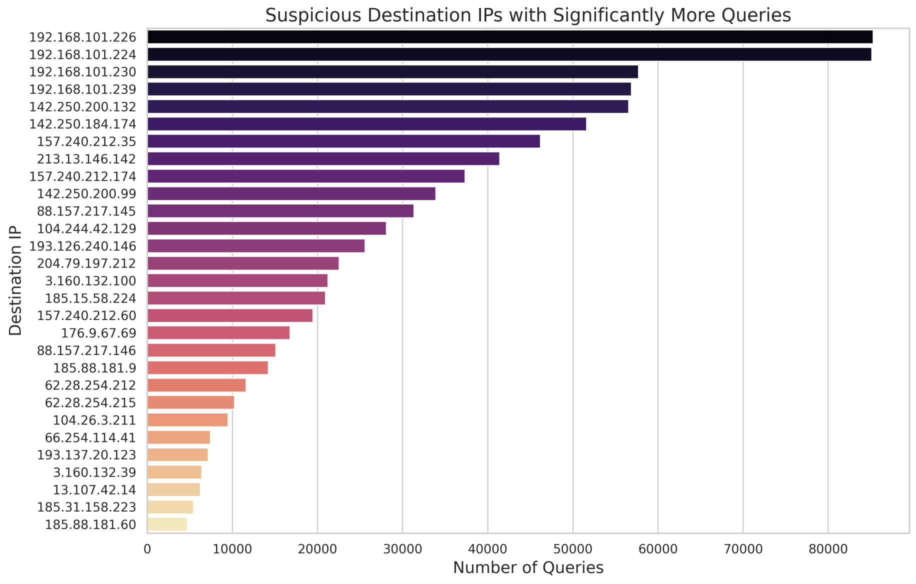

Figure 4.4: Top Destination IPs with Most Queries

Upon closer inspection, we can also observe that these IPs are part of the Top 10 Source-Destination IP Pairs with the Most DNS Queries 4.5.

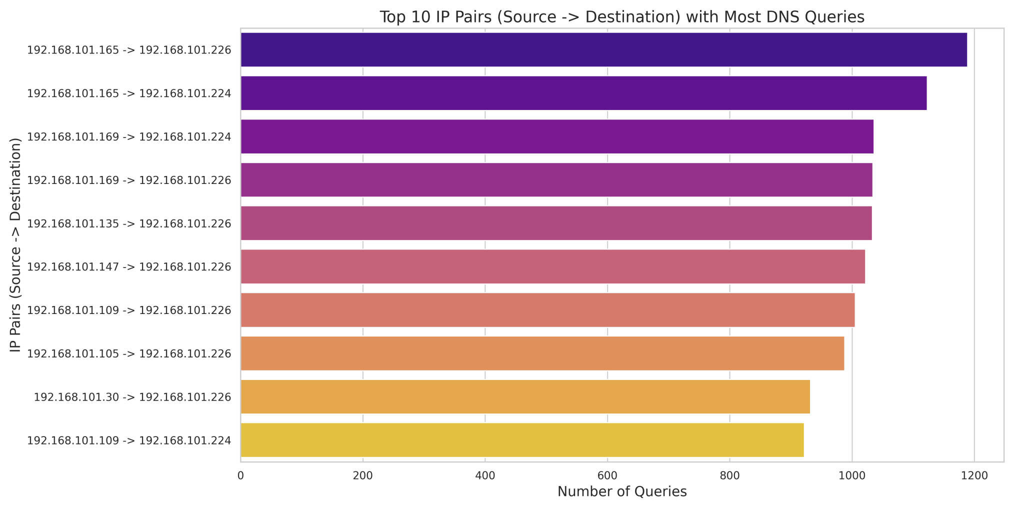

Figure 4.5: Top 10 Source IPs Querying Many DNS Servers

The significant increase in the number of flows directed towards IPs 224 and 228 may indicate that these servers are receiving commands or transferring data from a botnet, which is part of a coordinated attack.

We used this scripts to analyse the DNS flow and detect C&C attacks:

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pygeoip

sns.set(style="whitegrid")

# Define student numbers and calculate X
nmec1 = 103411
nmec2 = 103690
X = (nmec1 + nmec2) % 10

# File paths
datafile = f'./datasets/dataset{X}/data{X}.parquet'
geoip_db_country = './GeoIP_DBs/GeoIP.dat'
geoip_db_asn = './GeoIP_DBs/GeoIPASNum.dat'

# Read parquet data files
data = pd.read_parquet(datafile)

# Filtragem ajustada
dns_traffic_data = data[((data['proto'].str.lower() == 'udp'
    ) & (data['port'] == 53)) |
                        ((data['proto'].str.lower() == 'tcp'
                           ) & (data['port'] == 443))]
```

```python

if dns_traffic_data.empty:
    print("No DNS records found. Check the data and filtering.")
else:
    # Initialize GeoIP
    gi_country = pygeoip.GeoIP(geoip_db_country)
    gi_asn = pygeoip.GeoIP(geoip_db_asn)

    # Function to get country of an IP
    def get_country(ip):
        try:
            return gi_country.country_name_by_addr(ip) or 'Unknown'
        except Exception:
            return 'Unknown'

    # Function to get ASN of an IP
    def get_asn(ip):
        try:
            return gi_asn.org_by_addr(ip) or 'Unknown'
        except Exception:
            return 'Unknown'

    # Add country and ASN information
    dns_traffic_data['src_country'] = dns_traffic_data['src_ip'].apply(get_country)
    dns_traffic_data['dst_country'] = dns_traffic_data['dst_ip'].apply(get_country)
    dns_traffic_data['src_asn'] = dns_traffic_data['src_ip'].apply(get_asn)
    dns_traffic_data['dst_asn'] = dns_traffic_data['dst_ip'].apply(get_asn)

    # Function to save adjusted figures
    def save_fig_adjusted(filename, dpi=300):
        plt.savefig(filename, bbox_inches='tight', dpi=dpi)
        plt.close()

    # Count the number of queries per destination IP
    dst_ip_counts = dns_traffic_data['dst_ip'].value_counts()

    # Statistical analysis to find outliers
    mean_count = dst_ip_counts.mean()
    std_count = dst_ip_counts.std()
    threshold = mean_count + 3 * std_count  # Define a threshold as mean + 3 * standard deviation

    # Detect suspicious IPs
    suspicious_ips = dst_ip_counts[dst_ip_counts > threshold
        ].index.tolist()
    suspicious_counts = dst_ip_counts[dst_ip_counts >
        threshold]

    if suspicious_ips:
        print("Suspicious IPs detected:")
        for ip, count in suspicious_counts.items():
            print(f"IP: {ip}, Queries: {count}")
    else:
        print("No suspicious IPs detected.")

    # Visualize the suspicious IPs
    plt.figure(figsize=(12, 8))
    sns.barplot(x=suspicious_counts.values, y=
        suspicious_counts.index, palette='magma')
    plt.title("Suspicious Destination IPs with Significantly
         More Queries", fontsize=16)
    plt.xlabel("Number of Queries", fontsize=14)
    plt.ylabel("Destination IP", fontsize=14)
    save_fig_adjusted('suspicious_dst_ips.png')
```

This code is designed to identify suspicious IP addresses that exhibit unusually high DNS query activity. It begins by loading DNS traffic data from a Parquet file using the `pd.read_parquet` function, specifically targeting UDP traffic on port 53 and TCP traffic on port 443 to ensure only relevant DNS data is analyzed. This is done in the data loading and filtering section.

Next, the code enriches the data by adding country and ASN information for both the source and destination IP addresses using GeoIP databases. This is achieved by applying country and ASN lookups to each IP address using the `get_country` and `get_asn` functions.

After enriching the data, the code calculates the number of DNS queries directed to each destination IP address using the `value_counts` method. This helps quantify the volume of queries each IP has received. The code then calculates the average number of queries and the standard deviation across all destination IPs. By identifying IP addresses that have query counts significantly above the norm—those exceeding the mean by more than three standard deviations—the code flags these IPs as suspicious.

These outlier IPs are likely to indicate potential abnormal or malicious activity. Finally, it creates a bar chart to visualize these suspicious IP addresses 4.4.

This is the output oof the program:

```
Suspicious IPs detected:
IP: 192.168.101.226, Queries: 85266
IP: 192.168.101.224, Queries: 85136
IP: 192.168.101.230, Queries: 57718
IP: 192.168.101.239, Queries: 56875
IP: 142.250.200.132, Queries: 56554
```

```
IP: 142.250.184.174, Queries: 51604
IP: 157.240.212.35, Queries: 46182
IP: 213.13.146.142, Queries: 41416
IP: 157.240.212.174, Queries: 37350
IP: 142.250.200.99, Queries: 33894
IP: 88.157.217.145, Queries: 31333
IP: 104.244.42.129, Queries: 28103
IP: 193.126.240.146, Queries: 25609
IP: 204.79.197.212, Queries: 22558
IP: 3.160.132.100, Queries: 21238
IP: 185.15.58.224, Queries: 20958
IP: 157.240.212.60, Queries: 19456
IP: 176.9.67.69, Queries: 16755
IP: 88.157.217.146, Queries: 15102
IP: 185.88.181.9, Queries: 14251
IP: 62.28.254.212, Queries: 11595
IP: 62.28.254.215, Queries: 10263
IP: 104.26.3.211, Queries: 9478
IP: 66.254.114.41, Queries: 7422
IP: 193.137.20.123, Queries: 7175
IP: 3.160.132.39, Queries: 6404
IP: 13.107.42.14, Queries: 6246
IP: 185.31.158.223, Queries: 5455
IP: 185.88.181.60, Queries: 4743
```

4.4 External users using the corporate public services in an anomalous way

Our first step was to do a data filtering by focusing our analysis on external interactions with the corporate servers by creating the `external_accesses` contains only those rows from `servers_data` where `src_ip` is not part of the internal network and grouped access pattern by `rc_ip` and `dst_ip` with a count of occurrences `access_count`. Then we used the **IsonaltionForest** [2] to identify anomalies [1].

These were the anomalies detected:

```
Potential Anomalies:
             src_ip      dst_ip  access_count  anomaly_score
5    82.155.121.105  200.0.0.12         10386             -1
14   82.155.121.113  200.0.0.12           521             -1
29   82.155.121.127  200.0.0.12           606             -1
83   82.155.121.176  200.0.0.12          8577             -1
94   82.155.121.186  200.0.0.11          8330             -1
98    82.155.121.19  200.0.0.11         10591             -1
```

[1]IsolationForest is an unsupervised learning algorithm that detects anomalies by isolating them through an ensemble of "Isolation Trees," where the ease of isolating an observation indicates its anomaly level

```
105   82.155.121.196   200.0.0.12          426             -1
116   82.155.121.206   200.0.0.12         8881             -1
176    82.155.121.79   200.0.0.12         8425             -1
185    82.155.121.88   200.0.0.12          322             -1
```

These results are not enough to make a solid conclusion so, has mentioned in section 3.4, we inspected if these ips were also occurring outside of normal business hours.

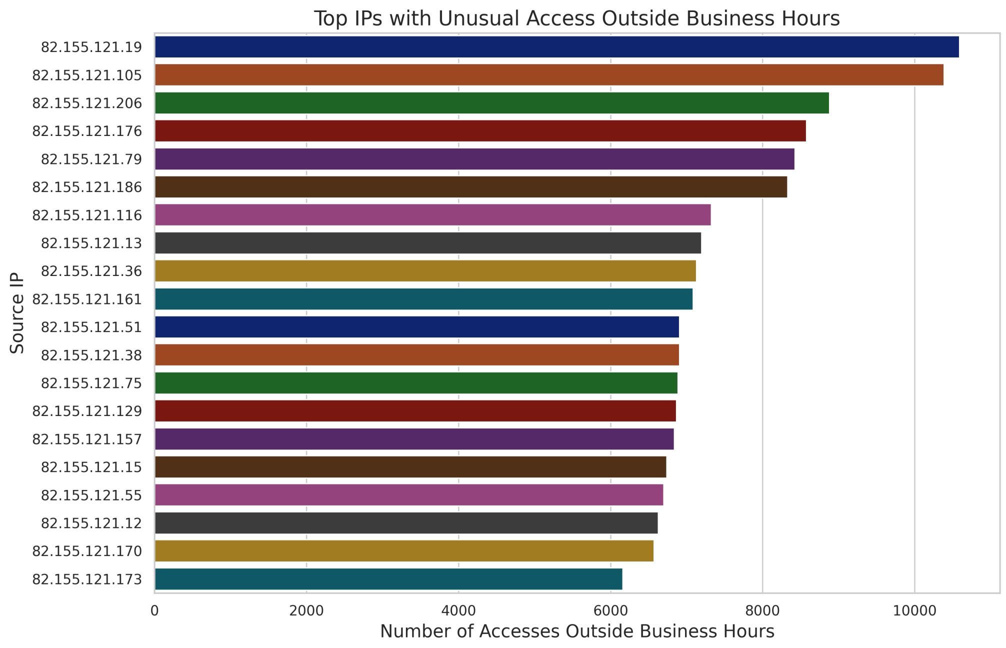

Figure 4.6: Top IPs with Unusual Access Outside Business Hours

If we merge these two evaluations we can can achieve that there is a match in the IPs 19 and 105 and say that these IPs are interacting with the corporation servers in an anomalous way.

To detect the anomaly was achieved with this script:

```
def detect_anomalies_in_external_accesses(servers_data):
    # Define the internal network of the corporation servers
    corporation_network = ipaddress.ip_network('200.0.0.0/24
        ')

    # Filter external accesses to the corporation servers
    external_accesses = servers_data[~servers_data['src_ip'
        ].apply(lambda ip: ipaddress.ip_address(ip) in
        corporation_network)]

    # Analyze access patterns (you can customize this based
        on your specific requirements)
    access_patterns = external_accesses.groupby(['src_ip', '
        dst_ip']).size().reset_index(name='access_count')

    # Use Isolation Forest for anomaly detection
    model = IsolationForest(contamination=0.1)  # Adjust
        contamination based on expected anomaly rate
    access_patterns['anomaly_score'] = model.fit_predict(
        access_patterns[['access_count']])

    # Identify potential anomalies
    potential_anomalies = access_patterns[access_patterns['
        anomaly_score'] == -1]

    return potential_anomalies

### EXECUTE ANALYSIS ------------------------
potential_anomalies = detect_anomalies_in_external_accesses(
    servers_data)

### PRINT POTENTIAL ANOMALIES ------------------------
print("Potential Anomalies:")
print(potential_anomalies)
```

To detect unusual access outside business hours was achieved using this script:

```
# Analyze access outside business hours
def analyze_public_service_access(access_data):
    access_data['timestamp'] = pd.to_datetime(access_data['
        timestamp'])
    access_data['hour'] = access_data['timestamp'].dt.hour

    out_of_hours = access_data[(access_data['hour'] < 8) | (
        access_data['hour'] > 18)]
    ip_counts = out_of_hours['src_ip'].value_counts().head
        (20)

    plt.figure(figsize=(12, 8))
    sns.barplot(x=ip_counts.values, y=ip_counts.index,
        palette='dark')
    plt.title("Top IPs with Unusual Access Outside Business
        Hours", fontsize=16)
    plt.xlabel("Number of Accesses Outside Business Hours",
        fontsize=14)
    plt.ylabel("Source IP", fontsize=14)
    save_fig_adjusted('top_out_of_hours_access.png')

    return ip_counts
```

It identifies and analyzes accesses to a public service that occur outside regular business hours (before 8 AM and after 6 PM). It processes a dataset to extract timestamps, determines the hour of each access, and filters out those that happen outside business hours. The function then counts the number of accesses from each unique IP address during these times and visualizes the top 20 IP addresses with the most accesses in a bar plot. Finally, it saves the plot and returns the counts of these top IP addresses.

Bibliography

[1] sosafe-awareness.com, *Top 5 cyber threats facing the public sector*, https://www.thinkdigitalpartners.com/news/2022/08/11/how-public-sector-organisations-can-protect-themselves-from-ddos-attacks/.

[2] scikit learn, *Isolationforest example*, https://scikit-learn.org/stable/auto_examples/ensemble/plot_isolation_forest.html.