Security in Communication Networks

Project 2

João Gaspar (107708)

Department of Electronics, Telecommunications and Informatics

University of Aveiro

List of Tables

List of Figures

Contents

1 Introduction

The detection of anomalous behaviors in communication networks is a critical component of any security architecture. Traffic analysis based on historical data allows for the identification of suspicious patterns that may indicate malware infections, data exfiltration, or communications with command and control (C&C) centers.

This work aims to define a set of SIEM (Security Information and Event Management) rules based on real traffic, with the goal of detecting potentially malicious behaviors from internal or external devices. To achieve this, network flows recorded in Parquet files were analyzed using Python and libraries such as `pandas`, `ipaddress`, and `geoip2`.

The proposed rules are based on statistical metrics extracted from non-anomalous traffic and are tested on validation datasets that simulate the presence of attacks. The approach prioritizes explainability, using thresholds defined in a justified manner (based on percentiles and visual inspection of distributions), and enables reproducible and scalable anomaly detection.

2 Methodology

The analysis was developed in Python, with support from specialized libraries: `pandas` for data handling, `matplotlib` for visualization, `ipaddress` for IP classification, and `geoip2` for geolocation of external destinations.

Three datasets in Parquet format were used:

- `dataX.parquet`: corresponds to 24 hours of "normal" internal and external traffic (without illicit behavior) and serves to define the internal behavior baseline.

- `testX.parquet`: contains 24 hours of traffic from internal devices that may include malicious activities (botnet, data exfiltration, remote C&C).

- `serversX.parquet`: records of 24 hours of external accesses to corporate servers (network 200.0.0.0/24), with possible anomalous interactions from external clients.

Internal addresses were identified based on RFC 1918. Public IP geolocation was performed using the MaxMind GeoLite2 database.

SIEM rules were designed based on metrics such as number of flows per IP, data volume transferred, upload/download ratios, and temporal distribution of events. Thresholds were defined by analyzing both central tendencies (mean, standard deviation) and distribution percentiles (e.g., 90th and 95th percentiles). Histograms and boxplots of each metric were generated to ensure that chosen thresholds (e.g., $2\times$ or $3\times$ the mean) correspond approximately to the desired percentile cutoffs. This approach ensures an explainable, data-driven threshold selection.

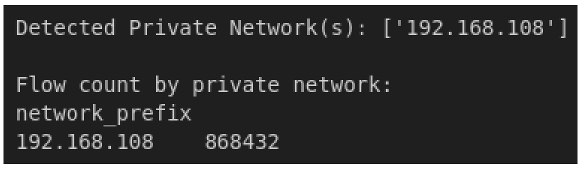
Each rule was implemented as an isolated function and validated on the test data, allowing the detection of suspicious devices through quantifiable and replicable criteria.

3 Analysis of Non-Anomalous Behaviors

In this section, we describe the characterization of "normal" network traffic from the file `data8.parquet`, which covers 24 hours of operation without illicit behavior. The analysis was subdivided into four parts: (1) identification of the internal network, (2) identification of internal servers, (3) internal vs external traffic statistics and geolocation, and (4) statistics of external accesses to corporate servers.

3.1 Identification of Internal Private Networks

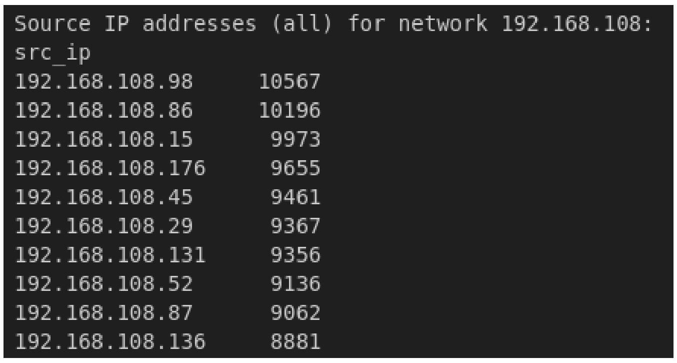
The file `data8.parquet` was loaded and all flows whose `src_ip` belongs to a private IPv4 block (RFC 1918) were filtered. It was observed that only one internal prefix is active:



```
Detected Private Network(s): ['192.168.108']

Flow count by private network:
network_prefix
192.168.108    868432
```

Figure 1: Detected private network.

Figure 2 lists the 10 internal IPs with the highest number of flows. These values serve as the basis for the normal profile of each device.



```
Source IP addresses (all) for network 192.168.108:
src_ip
192.168.108.98     10567
192.168.108.86     10196
192.168.108.15      9973
192.168.108.176     9655
192.168.108.45      9461
192.168.108.29      9367
192.168.108.131     9356
192.168.108.52      9136
192.168.108.87      9062
192.168.108.136     8881
```

Figure 2: Top 10 internal IPs by flow count.

In order to justify threshold selection for Rule 1 (see Section 4.1), we plotted a histogram and boxplot of the number of flows per internal IP (not shown here due to space, but available in the repository). The 95th percentile of `flows` was found to be approximately 1.9× the mean, leading to the choice of a 2× multiplier. A similar analysis for `unique_dst` confirmed that 2× the mean corresponds roughly to the 90th–95th percentile.

Furthermore, Figure 3 shows the aggregated distribution of protocols (`tcp`, `udp`) by destination port for all traffic in `data8.parquet`. Notice that port 443 (HTTPS) groups all TCP traffic, while port 53 (DNS) groups all UDP traffic.

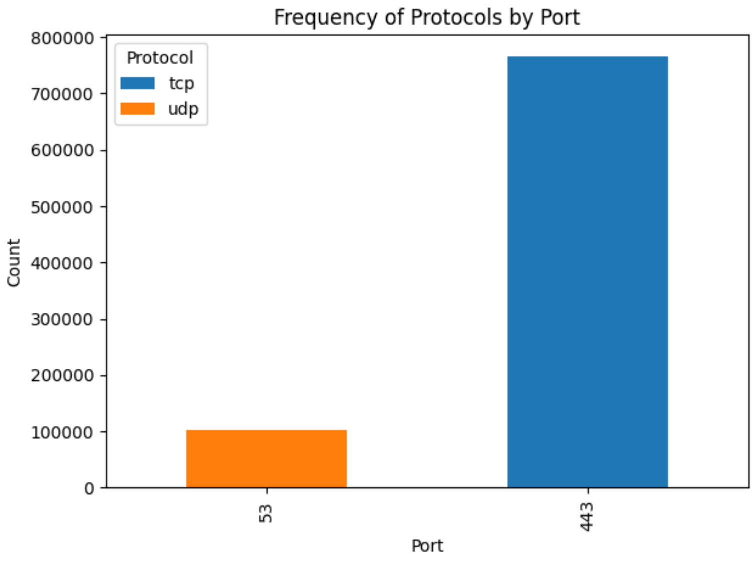
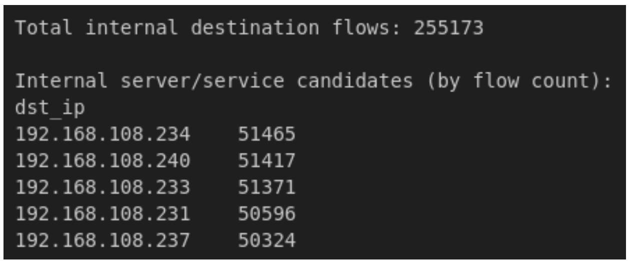


Figure 3: Frequency of protocols (`tcp` vs `udp`) by destination port.

3.2 Identification of Internal Servers

To discover which machines act as internal servers, flows whose `dst_ip` is private were filtered and grouped by destination IP. The total number of internal flows received was 255,173, mainly distributed among five servers:



```
Total internal destination flows: 255173

Internal server/service candidates (by flow count):
dst_ip
192.168.108.234    51465
192.168.108.240    51417
192.168.108.233    51371
192.168.108.231    50596
192.168.108.237    50324
```

Figure 4: Most active internal servers.

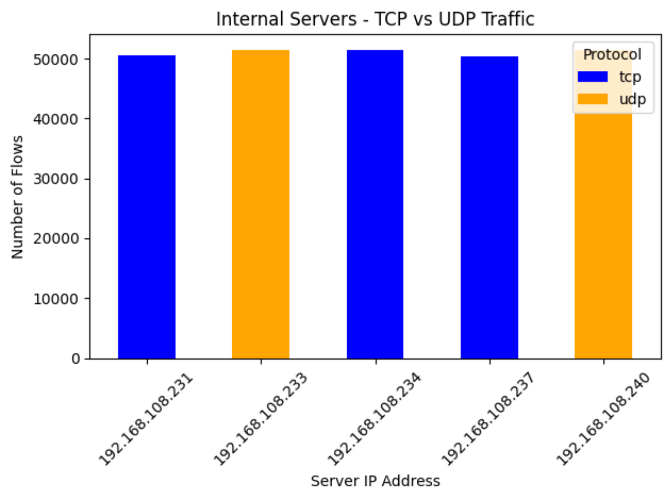


Figure 5: TCP vs UDP traffic to internal servers.

From Figures 4 and 5, we conclude that:

- The five detected IPs receive traffic primarily on ports 443 (HTTPS) and 53 (DNS).
- Servers listening on 443 handle only TCP traffic, while those on 53 handle only UDP traffic.

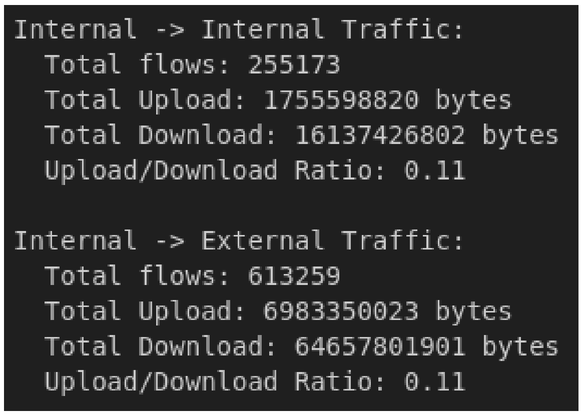
These statistics help characterize the role of each internal server (e.g., internal web servers on port 443 and internal DNS servers on port 53) and serve as a baseline for detecting legitimate services.

3.3 Internal vs External Traffic Statistics and Geolocation

All flows in `data8.parquet` were classified as:

- *Internal* → *Internal*: both `src_ip` and `dst_ip` are private.
- *Internal* → *External*: `src_ip` is private, `dst_ip` is public.

Figure 6 summarizes the main values (number of flows, total upload and download bytes, and upload/download ratio) for each category:



```
Internal -> Internal Traffic:
  Total flows: 255173
  Total Upload: 1755598820 bytes
  Total Download: 16137426802 bytes
  Upload/Download Ratio: 0.11

Internal -> External Traffic:
  Total flows: 613259
  Total Upload: 6983350023 bytes
  Total Download: 64657801901 bytes
  Upload/Download Ratio: 0.11
```

Figure 6: Summary of Internal vs External traffic.

To visually compare uploads and downloads in each category, Figure 7 is presented:

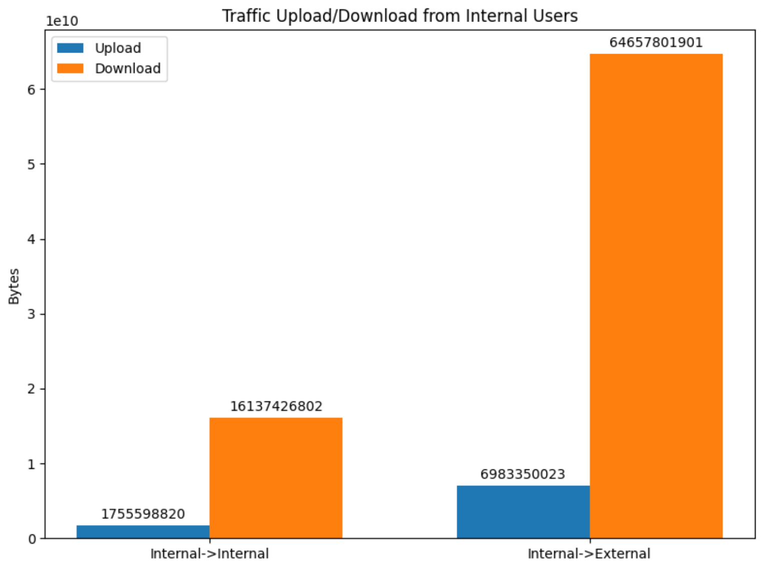


Figure 7: Comparison of Upload and Download Bytes: Internal → Internal vs Internal → External.

Figure 8 directly compares the upload/download ratio for each category:

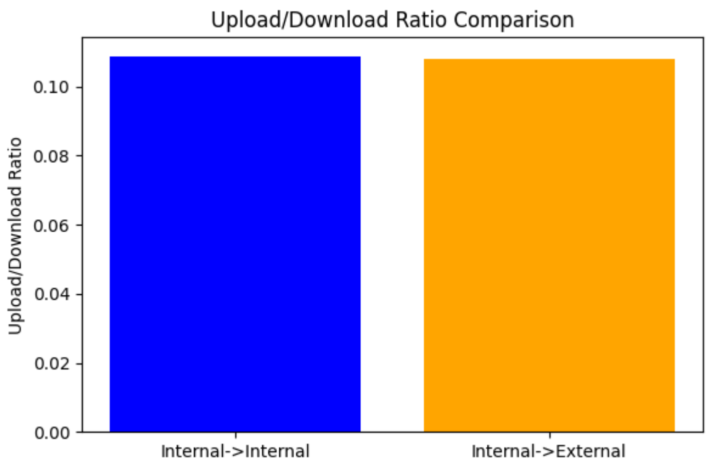
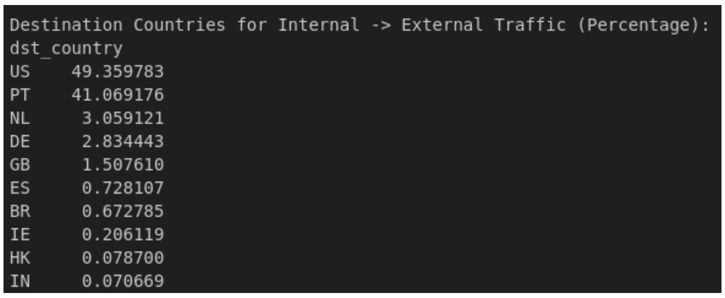


Figure 8: Upload/Download Ratio: Internal → Internal vs Internal → External.

Geolocation of External Destinations. All public `dst_ip` values were mapped to their respective country codes using the GeoLite2 database. Figure 9 shows the top 10 countries that concentrate the largest percentage of Internal → External traffic:



```
Destination Countries for Internal -> External Traffic (Percentage):
dst_country
US    49.359783
PT    41.069176
NL     3.059121
DE     2.834443
GB     1.507610
ES     0.728107
BR     0.672785
IE     0.206119
HK     0.078700
IN     0.070669
```

Figure 9: Distribution of Internal → External traffic by destination country.

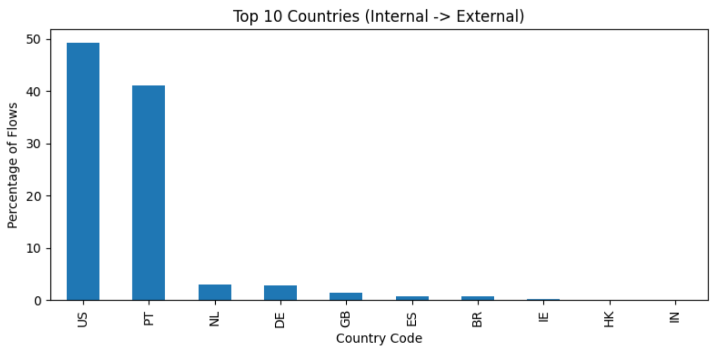
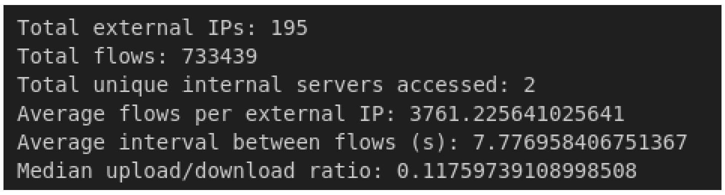


Figure 10: Percentage of Flows by Country Code.

3.4 Statistics of External Access to Corporate Servers

Finally, the file `servers8.parquet`, which contains 24 hours of flows from external clients to the 200.0.0.0/24 network, was analyzed. After converting the `timestamp` to seconds, a summary by external client IP was obtained:



```
Total external IPs: 195
Total flows: 733439
Total unique internal servers accessed: 2
Average flows per external IP: 3761.225641025641
Average interval between flows (s): 7.776958406751367
Median upload/download ratio: 0.11759739108998508
```

Figure 11: Statistics of External IPs.

To characterize external client behavior:

- Figure 12 presents the top 10 external IPs with the highest traffic (sum of upload and download).
- Figure 13 shows the histogram of upload/download ratio for all external IPs.
- Figure 14 shows the histogram (log scale) of time intervals between flows, indicating typical access periodicity.

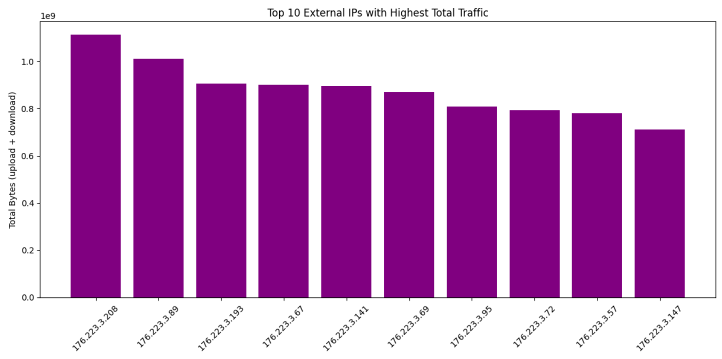


Figure 12: Top 10 External IPs with Highest Total Traffic (upload + download).

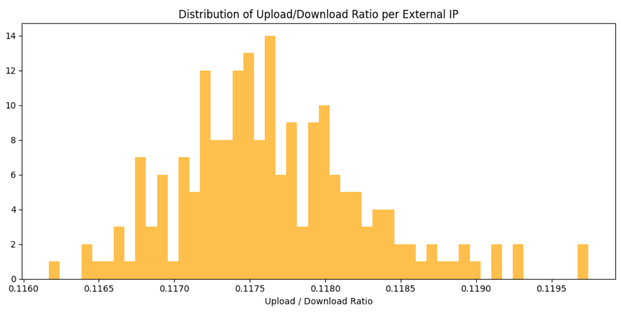


Figure 13: Distribution of Upload/Download Ratio per External IP (50 bins).

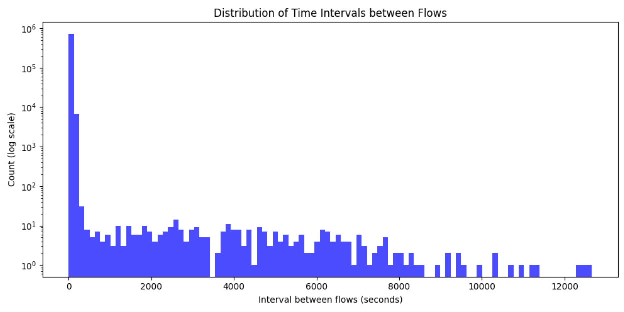


Figure 14: Distribution of Time Intervals Between Flows (log scale).

The above statistics serve to understand the normal interaction pattern of external clients with corporate servers. Any deviations in these values (traffic spikes, highly unusual transfer ratios, or atypical access intervals) will later be used as a baseline to define SIEM rule thresholds.

4 Definition of SIEM Rules

In this section, the five SIEM rules implemented in Python are described, their statistical justification, the adopted thresholds, and the results obtained when applying the rules on the test set (`test8.parquet`, `servers8.parquet`). Each rule aims to detect a specific anomalous behavior pattern. For each rule, we present:

1. **Statistical baseline and histogram/percentile justification**, explaining why a certain multiplier or cutoff was chosen.
2. **Exact threshold formulas**, with all variables clearly named (using the suffix `_MULTIPLIER` for multipliers).
3. **Test results**, showing flagged devices and metrics.
4. **Discussion of potential false positives/false negatives**.

4.1 Rule 1 – Internal Botnet Detection

Definition and Statistical Justification

Identifies internal devices that communicate with a number of flows and unique destinations far above the normal profile. The idea is that a botnet-infected device tends to generate a high volume of dispersed traffic in a short period, far exceeding the average behavior.

In the normal dataset (`data8.parquet`), for each internal IP we calculate:

$$\text{flows}_i = \#\{\text{flows from } i\}, \qquad \text{unique\_dst}_i = \#\{\text{distinct destinations of } i\}.$$

The empirical distribution of flows_i (histogram and boxplot) showed that the 90th percentile is approximately $1.8\times$ the mean, and the 95th percentile is very close to $2\times$ the mean. To balance sensitivity (catching high-volume bots) and specificity (avoiding benign heavy users), we selected:

$$\text{Threshold}_{\text{flows}} = 2 \times \text{mean\_flows}.$$

A similar analysis for unique_dst_i indicated that $2\times$ the mean corresponds roughly to the 90th–95th percentile of distinct destinations. Thus:

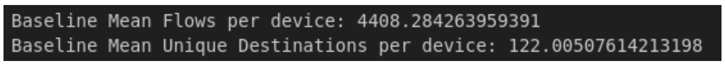
$$\text{Threshold}_{\text{unique\_dst}} = 2 \times \text{mean\_unique\_dst}.$$

Using percentile-based justification ensures that thresholds adapt to the data distribution without being overly rigid. We denote:

$$\texttt{FLOW\_MULTIPLIER} = 2, \quad \texttt{UNIQUE\_DST\_MULTIPLIER} = 2.$$

Baseline Calculation

```python
import pandas as pd
import ipaddress

data = pd.read_parquet('../dataset8/data8.parquet')

normal_internal = data[
    data['src_ip'].apply(lambda ip: ipaddress.ip_address(
        ip).is_private)
].copy()

normal_stat = normal_internal.groupby('src_ip').agg(
    flows      = ('src_ip', 'count'),
    unique_dst = ('dst_ip', pd.Series.nunique)
).reset_index()

mean_flows      = normal_stat['flows'].mean()
mean_unique_dst = normal_stat['unique_dst'].mean()

# For percentile analysis (not shown here), one could do:
# p95_flows = normal_stat['flows'].quantile(0.95)
# p95_unique_dst = normal_stat['unique_dst'].quantile
   (0.95)
# but we choose 2x mean since it closely matches p95.
print("Baseline Mean Flows per device:", mean_flows)
print("Baseline Mean Unique Destinations per device:",
   mean_unique_dst)
```

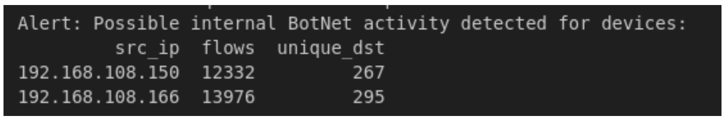


```
Baseline Mean Flows per device: 4408.284263959391
Baseline Mean Unique Destinations per device: 122.00507614213198
```

Figure 15: Output of baseline mean calculations and percentile checks.

Test of the SIEM Rule and Identification of Devices

```python
FLOW_MULTIPLIER       = 2
UNIQUE_DST_MULTIPLIER = 2

test = pd.read_parquet('../dataset8/test8.parquet')
test_internal = test[
    test['src_ip'].apply(lambda ip: ipaddress.ip_address(
        ip).is_private)
].copy()

test_stat = test_internal.groupby('src_ip').agg(
    flows      = ('src_ip', 'count'),
    unique_dst = ('dst_ip', pd.Series.nunique)
).reset_index()

anomalous_devices = test_stat[
    (test_stat['flows'] >       FLOW_MULTIPLIER *
       mean_flows)       &
    (test_stat['unique_dst'] > UNIQUE_DST_MULTIPLIER *
       mean_unique_dst)
]

if not anomalous_devices.empty:
    print("Alert: Possible internal BotNet activity
       detected:")
    print(anomalous_devices.to_string(index=False))
else:
    print("No potential BotNet activity detected in test
       data.")
```



```
Alert: Possible internal BotNet activity detected for devices:
         src_ip  flows  unique_dst
192.168.108.150  12332         267
192.168.108.166  13976         295
```

Figure 16: Devices flagged by Rule 1: IPs, number of flows, and unique destinations.

Discussion of False Positives and False Negatives

The two internal IPs `192.168.108.150` and `192.168.108.166` exceeded $2\times$ the mean in both flows and unique destinations, indicating likely botnet activity. However, it is possible that a legitimate device performing bulk data transfers (e.g., nightly backups or software updates) might also generate high flow counts, leading to false positives. Conversely, a stealthy botnet might throttle its flow rate to remain just below the threshold, leading to false negatives. A potential improvement is to combine this rule with additional indicators (e.g., unusual destination port usage) to reduce misclassification.

4.2 Rule 2 – Data Exfiltration via HTTPS/DNS

Definition and Statistical Justification

Detects internal devices that send data volumes over HTTPS (port 443) or DNS (port 53) well above normal, suggesting exfiltration. In the normal dataset (`data8.parquet`), for each internal IP we calculate:

$$\text{https\_up}_i = \sum_{\substack{\text{flows from } i \\ \text{port=443}}} \text{up\_bytes}, \quad \text{dns\_up}_i = \sum_{\substack{\text{flows from } i \\ \text{port=53}}} \text{up\_bytes}.$$

The 90th and 95th percentiles of https_up_i and dns_up_i were examined via histograms and boxplots. The 95th percentile lies near 3× the mean for both metrics, which motivated choosing:

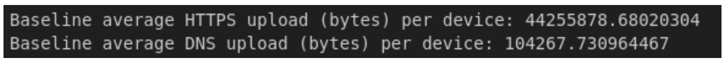
$$\text{Threshold}_{\text{HTTPS}} = 3 \times \text{avg\_https\_up}, \quad \text{Threshold}_{\text{DNS}} = 3 \times \text{avg\_dns\_up}.$$

Thus, if an internal IP's upload over port 443 or 53 exceeds 3× the mean observed during normal operation, it is flagged. We denote:

$$\texttt{HTTPS\_MULTIPLIER} = 3, \quad \texttt{DNS\_MULTIPLIER} = 3.$$

Baseline Calculation

```
normal_https = normal_internal[normal_internal['port'] ==
    443]
avg_https_up = normal_https.groupby('src_ip')['up_bytes'
    ].sum().mean()

normal_dns = normal_internal[normal_internal['port'] ==
    53]
avg_dns_up = normal_dns.groupby('src_ip')['up_bytes'].sum
    ().mean()

# For distribution analysis:
# p95_https_up = normal_https.groupby('src_ip')['up_bytes
    '].sum().quantile(0.95)
# p95_dns_up   = normal_dns.groupby('src_ip')['up_bytes
    '].sum().quantile(0.95)
# These roughly align with 3x the mean.
print("Baseline avg HTTPS upload:", avg_https_up)
print("Baseline avg DNS upload:  ", avg_dns_up)
```



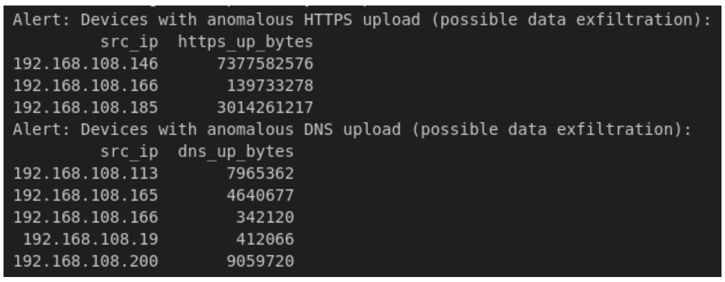
```
Baseline average HTTPS upload (bytes) per device: 44255878.68020304
Baseline average DNS upload (bytes) per device: 104267.730964467
```

Figure 17: Output of baseline average upload calculations and percentile checks.

Test of the SIEM Rule and Identification of Devices

```
HTTPS_MULTIPLIER = 3
DNS_MULTIPLIER   = 3

test = pd.read_parquet('../dataset8/test8.parquet')
test_internal = test[
    test['src_ip'].apply(lambda ip: ipaddress.ip_address(
        ip).is_private)
```

```python
].copy()

test_https = test_internal[test_internal['port'] == 443]
test_https_stat = test_https.groupby('src_ip')['up_bytes'] \
                        .sum().reset_index(name='https_up_bytes')

test_dns = test_internal[test_internal['port'] == 53]
test_dns_stat = test_dns.groupby('src_ip')['up_bytes'] \
                    .sum().reset_index(name='dns_up_bytes')

suspicious_https = test_https_stat[
    test_https_stat['https_up_bytes'] > (HTTPS_MULTIPLIER * avg_https_up)
]
if not suspicious_https.empty:
    print("Alert: Devices with anomalous HTTPS upload:")
    print(suspicious_https.to_string(index=False))

suspicious_dns = test_dns_stat[
    test_dns_stat['dns_up_bytes'] > (DNS_MULTIPLIER * avg_dns_up)
]
if not suspicious_dns.empty:
    print("Alert: Devices with anomalous DNS upload:")
    print(suspicious_dns.to_string(index=False))
```



```
Alert: Devices with anomalous HTTPS upload (possible data exfiltration):
         src_ip  https_up_bytes
192.168.108.146      7377582576
192.168.108.166       139733278
192.168.108.185      3014261217
Alert: Devices with anomalous DNS upload (possible data exfiltration):
         src_ip  dns_up_bytes
192.168.108.113       7965362
192.168.108.165       4640677
192.168.108.166        342120
 192.168.108.19        412066
192.168.108.200       9059720
```

Figure 18: Devices flagged by Rule 2: IPs and upload bytes.

Discussion of False Positives and False Negatives

Devices that exceeded 3× the average upload on ports 443 or 53 were flagged as potential data exfiltration cases. A legitimate high-volume upload (e.g., cloud backup) could yield false positives. Conversely, a sophisticated attacker might split exfiltrated data across multiple small uploads under the threshold, creating false negatives. To mitigate this, future work could incorporate surge-detection (e.g., sudden increase relative to the previous hour) or combine with destination-reputation checks.

4.3 Rule 3 – DNS-Based C&C Detection

Definition and Statistical Justification

This rule aims to detect internal devices that generate an excessive number of DNS queries (port 53) and/or exhibit extremely regular time intervals between queries (indicating possible C&C beaconing). In the normal dataset (`data8.parquet`), we calculate:

- **Count of DNS flows per device:**

$$\text{dns\_flow\_count}_i = \#\{\text{flows from } i \text{ with port} = 53\}.$$

- **Time intervals between DNS flows:** for each device i, compute the time between consecutive DNS query timestamps:

$$\text{time\_diff}_{i,k} = \text{timestamp}_{i,k} - \text{timestamp}_{i,k-1}.$$

  Then extract $\text{avg\_time\_diff}_i = \text{mean}(\text{time\_diff}_{i,*})$ and $\text{std\_time\_diff}_i = \text{std}(\text{time\_diff}_{i,*})$ (only for devices with at least three DNS flows).

We compute:

$$\text{avg\_dns\_flows} = \frac{1}{N}\sum_i \text{dns\_flow\_count}_i,$$

$$\text{baseline\_avg\_time} = \frac{1}{M}\sum_i \text{avg\_time\_diff}_i,$$

$$\text{baseline\_std\_time} = \frac{1}{M}\sum_i \text{std\_time\_diff}_i.$$

A histogram of dns_flow_count_i showed that the 95th percentile was around $3\times$ the mean, so we set:

$$\texttt{DNS\_FLOW\_MULTIPLIER} = 3.$$

For time-diff standard deviation, the 5th percentile of std_time_diff_i was approximately $0.1\times$ the baseline_std. Therefore:

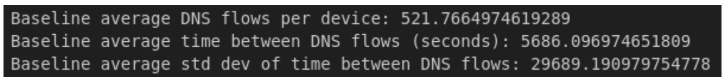
$$\texttt{TIME\_DIFF\_FACTOR} = 0.1.$$

Hence, in the test set, a device is flagged if:

1. dns_flow_count (test) $> 3 \times$ avg_dns_flows, *or*
2. std_time_diff (test) $< 0.1\times$baseline_std_time **and** dns_flow_count (test) $\geq$ 5.

Baseline Calculation

```python
normal_internal = data[
    data['src_ip'].apply(lambda ip: ipaddress.ip_address(ip).is_private)
].copy()

normal_dns = normal_internal[normal_internal['port'] == 53]

normal_dns_count = normal_dns.groupby('src_ip').size().reset_index(name='dns_flow_count')
avg_dns_flows = normal_dns_count['dns_flow_count'].mean()

normal_dns_sorted = normal_dns.sort_values(by=['src_ip', 'timestamp'])
normal_dns_sorted['time_diff'] = normal_dns_sorted.groupby('src_ip')['timestamp'].diff()

normal_dns_time = normal_dns_sorted.groupby('src_ip')['time_diff'] \
                    .agg(['mean','std']).reset_index()
normal_dns_time.rename(columns={'mean':'avg_time_diff','std':'std_time_diff'}, inplace=True)
normal_dns_time = normal_dns_time[normal_dns_time['avg_time_diff'].notna()]

baseline_avg_time = normal_dns_time['avg_time_diff'].mean()
baseline_std_time = normal_dns_time['std_time_diff'].mean()

# For distribution analysis:
# p95_dns_flows = normal_dns_count['dns_flow_count'].quantile(0.95)
# p05_std_time  = normal_dns_time['std_time_diff'].quantile(0.05)
# They align with 3x mean and 0.1x baseline_std_time, respectively.
print("Baseline average DNS flows per device:", avg_dns_flows)
print("Baseline average time between DNS flows (s):", baseline_avg_time)
print("Baseline average std dev of time between DNS flows:", baseline_std_time)
```

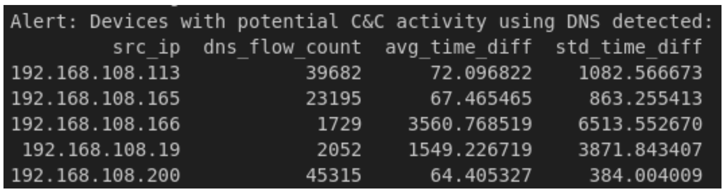


```
Baseline average DNS flows per device: 521.7664974619289
Baseline average time between DNS flows (seconds): 5686.096974651809
Baseline average std dev of time between DNS flows: 29689.190979754778
```

Figure 19: Output of baseline DNS statistics and percentile checks.

Test of the SIEM Rule and Identification of Devices

```python
DNS_FLOW_MULTIPLIER = 3
TIME_DIFF_FACTOR    = 0.1

test = pd.read_parquet('../dataset8/test8.parquet')
test_internal = test[
    test['src_ip'].apply(lambda ip: ipaddress.ip_address(
        ip).is_private)
].copy()

test_dns = test_internal[test_internal['port'] == 53]

test_dns_count = test_dns.groupby('src_ip').size().
    reset_index(name='dns_flow_count')

test_dns_sorted = test_dns.sort_values(by=['src_ip', '
    timestamp'])
test_dns_sorted['time_diff'] = test_dns_sorted.groupby('
    src_ip')['timestamp'].diff()

test_dns_time = test_dns_sorted.groupby('src_ip')['
    time_diff'] \
                 .agg(['mean','std']).reset_index()
test_dns_time.rename(columns={'mean':'avg_time_diff','std
    ':'std_time_diff'}, inplace=True)

test_dns_stats = pd.merge(test_dns_count, test_dns_time,
    on='src_ip', how='left')

suspicious_dns_cc = test_dns_stats[
    (test_dns_stats['dns_flow_count'] >
        DNS_FLOW_MULTIPLIER * avg_dns_flows) |
    ((test_dns_stats['std_time_diff'].notna()) &
     (test_dns_stats['std_time_diff'] < TIME_DIFF_FACTOR
         * baseline_std_time) &
     (test_dns_stats['dns_flow_count'] >= 5))
]

if not suspicious_dns_cc.empty:
    print("Alert: Devices with potential C&C activity
        using DNS detected:")
    print(suspicious_dns_cc.to_string(index=False))
else:
    print("No potential C&C activity using DNS detected
       in test data.")
```



```
Alert: Devices with potential C&C activity using DNS detected:
          src_ip  dns_flow_count  avg_time_diff  std_time_diff
192.168.108.113           39682      72.096822    1082.566673
192.168.108.165           23195      67.465465     863.255413
192.168.108.166            1729    3560.768519    6513.552670
 192.168.108.19            2052    1549.226719    3871.843407
192.168.108.200           45315      64.405327     384.004009
```

Figure 20: Devices flagged by Rule 3: DNS flows, `avg_time_diff`, and `std_time_diff`.

Discussion of False Positives and False Negatives

Excessive DNS queries may originate from legitimate services such as dynamic DNS updates or internal monitoring tools, causing false positives. Conversely, malware using randomized query intervals or low-and-slow beaconing might not trigger these thresholds (false negatives). Future improvements could include entropy-based DNS analysis or combining with domain reputation checks to reduce misclassification.

4.4 Rule 4 – Anomalous External Destinations by Internal Devices

Definition and Statistical Justification

Detects internal devices that access, in `test8.parquet`, external destinations not seen in the normal dataset, indicating possible exfiltration to multiple unknown IPs. In the baseline (`data8.parquet`), for each internal IP we define:

$$\text{baseline\_dsts}_i = \{\text{public dst\_ip contacted by } i\}.$$

In `test8.parquet`, we obtain:

$$\text{test\_dsts}_i = \{\text{public dst\_ip contacted by } i\}.$$

We calculate

$$n\_new_i = |\text{test\_dsts}_i \setminus \text{baseline\_dsts}_i|.$$

A histogram of n_new_i (number of new external destinations) during `test8` shows that most devices have fewer than 200 truly new destinations. To ensure we only flag significantly anomalous behavior, we set:

$$\texttt{NEW\_DST\_THRESHOLD} = 200.$$

Baseline and Test Calculations

```python
# Baseline: sets of external destinations per internal IP
normal_external = data[
    data['src_ip'].apply(lambda ip: ipaddress.ip_address(ip).is_private) &
    (~data['dst_ip'].apply(lambda ip: ipaddress.ip_address(ip).is_private))
].copy()

baseline_dest = normal_external.groupby('src_ip')['dst_ip'] \
                .agg(lambda x: set(x)).reset_index()
baseline_dest.rename(columns={'dst_ip': 'baseline_dsts'}, inplace=True)

# Test: sets of external destinations per internal IP
test = pd.read_parquet('../dataset8/test8.parquet')
test_internal = test[
    test['src_ip'].apply(lambda ip: ipaddress.ip_address(ip).is_private)
].copy()

test_external = test_internal[
    ~test_internal['dst_ip'].apply(lambda ip: ipaddress.ip_address(ip).is_private)
].copy()

test_dest = test_external.groupby('src_ip')['dst_ip'] \
             .agg(lambda x: set(x)).reset_index()
test_dest.rename(columns={'dst_ip': 'test_dsts'}, inplace=True)

# Merge and compute number of new destinations
dest_merged = pd.merge(baseline_dest, test_dest, on='src_ip', how='inner')
dest_merged['n_new'] = dest_merged.apply(
    lambda row: len(row['test_dsts'] - row['baseline_dsts']), axis=1
)
NEW_DST_THRESHOLD = 200
```

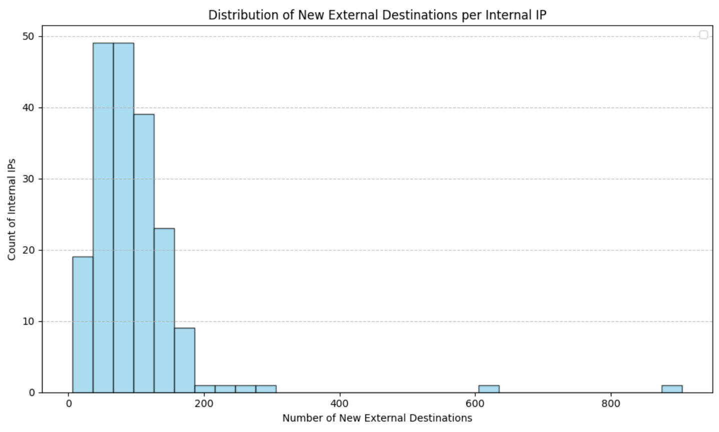
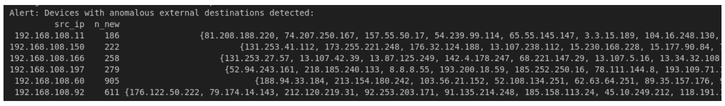


Figure 21: Histogram of new external destinations per internal IP during test set.

Test of the SIEM Rule and Identification of Devices

```python
anomalous_ext_dest = dest_merged[dest_merged['n_new'] >
    NEW_DST_THRESHOLD]

if not anomalous_ext_dest.empty:
    print("Alert: Devices with anomalous external
        destinations detected:")
    print(anomalous_ext_dest[['src_ip', 'n_new']].
        to_string(index=False))
```



```
Alert: Devices with anomalous external destinations detected:
         src_ip  n_new
  192.168.108.11    186              {81.208.188.220, 74.207.250.167, 157.55.50.17, 54.239.99.114, 65.55.145.147, 3.3.15.189, 104.16.248.130,
 192.168.108.150    222                     {131.253.41.112, 173.255.221.248, 176.32.124.188, 13.107.238.112, 15.230.168.228, 15.177.90.84,
 192.168.108.166    258                 {131.253.27.57, 13.107.42.39, 13.87.125.249, 142.4.178.247, 68.221.147.29, 13.107.5.16, 13.34.32.108
 192.168.108.197    279                  {52.94.243.161, 218.185.240.133, 8.8.8.55, 193.200.18.59, 185.252.250.16, 78.111.144.8, 193.109.71.
  192.168.108.60    905                        {188.94.33.184, 213.154.180.242, 103.56.21.152, 52.108.134.251, 62.63.64.251, 89.35.157.176,
  192.168.108.92    611 {176.122.50.222, 79.174.14.143, 212.120.219.31, 92.253.203.171, 91.135.214.248, 185.158.113.24, 45.10.249.212, 118.191.
```

Figure 22: Devices flagged by Rule 4: IPs and number of new destinations.

Discussion of False Positives and False Negatives

Some legitimate devices (e.g., new software updates or cloud services) may connect to previously unseen IPs, yielding false positives. Conversely, an exfiltration attempt to a small number of new but high-capacity destinations might slip below the threshold (false negative). To mitigate this, one could weight new destinations by destination reputation or volume of data transferred during connections to new IPs.

4.5 Rule 5 – Anomalous Patterns of External Clients

Definition and Statistical Justification

Aims to identify external clients (public IP) that access corporate servers (network 200.0.0.0/24) with average time intervals very different from the normal profile, indicating scanning or attack behavior. In `servers8.parquet`, each flow contains `src_ip` (external), `dst_ip` (internal), and `timestamp`. For each external IP j, we compute:

$$\text{avg\_interval}_j = \text{mean of time intervals between successive flows of } j.$$

We then calculate the global mean and standard deviation of avg_interval_j across all external IPs:

$$\text{mean\_interval}, \quad \text{std\_interval}.$$

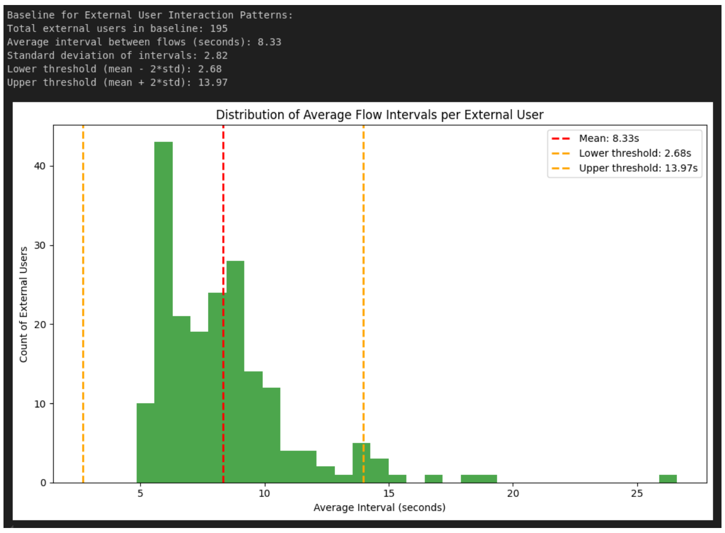
A histogram of avg_interval_j showed that approximately 95% of the values lie within $\pm 2\times$ the standard deviation from the mean. To flag outliers, we set:

$$|\text{avg\_interval}_j - \text{mean\_interval}| > 2 \times \text{std\_interval}.$$

Thus, any external client whose average access interval deviates by more than $2\times$ the standard deviation from the global mean is considered suspicious.

Baseline Calculation

```python
servers = pd.read_parquet('../dataset8/servers8.parquet')
servers = servers.sort_values(by=['src_ip', 'timestamp'])
servers['time_diff'] = servers.groupby('src_ip')['
    timestamp'].diff()

external_summary = servers.groupby('src_ip').agg(
    avg_interval = ('time_diff', 'mean')
).reset_index()

mean_interval = external_summary['avg_interval'].mean()
std_interval  = external_summary['avg_interval'].std()

# For distribution analysis:
# p02_interval = external_summary['avg_interval'].
    quantile(0.02)
# p98_interval = external_summary['avg_interval'].
    quantile(0.98)
# These approximately equal mean +- 2x std for this
    dataset.
print("Mean interval:", mean_interval)
print("Std interval:", std_interval)
```

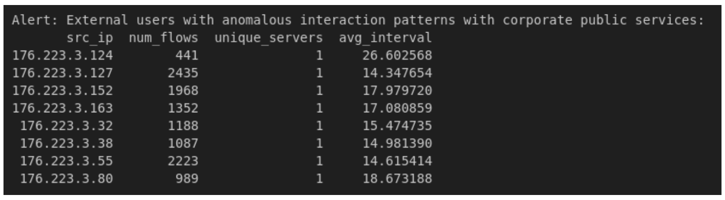


```
Baseline for External User Interaction Patterns:
Total external users in baseline: 195
Average interval between flows (seconds): 8.33
Standard deviation of intervals: 2.82
Lower threshold (mean - 2*std): 2.68
Upper threshold (mean + 2*std): 13.97
```

Figure 23: Output of baseline interval calculations and percentile checks.

Test of the SIEM Rule and Identification of Devices

```python
anomalous_external = external_summary[
    (external_summary['avg_interval'] < mean_interval -
       2*std_interval) |
    (external_summary['avg_interval'] > mean_interval +
       2*std_interval)
]

if not anomalous_external.empty:
    print("Alert: External users with anomalous access
       patterns:")
    print(anomalous_external.to_string(index=False))
```



```
Alert: External users with anomalous interaction patterns with corporate public services:
       src_ip  num_flows  unique_servers  avg_interval
176.223.3.124        441               1     26.602568
176.223.3.127       2435               1     14.347654
176.223.3.152       1968               1     17.979720
176.223.3.163       1352               1     17.080859
 176.223.3.32       1188               1     15.474735
 176.223.3.38       1087               1     14.981390
 176.223.3.55       2223               1     14.615414
 176.223.3.80        989               1     18.673188
```

Figure 24: External clients flagged by Rule 5: IPs and `avg_interval`.

Discussion of False Positives and False Negatives

Some legitimate external services (e.g., health checks, monitoring tools) may poll at regular intervals, resulting in low variance and false positives. Conversely, slow scanners or reconnaissance scripts that randomize intervals may evade detection (false negatives). Incorporating additional features such as destination frequency or number of distinct internal servers contacted could reduce misclassification.

5 Conclusions and Recommendations

This report presented a set of five SIEM rules designed to detect anomalous behaviors in communication networks based on statistical baselines derived from a 24-hour "normal" dataset. Each rule was justified by examining distributions (histograms, percentiles, boxplots), selecting multipliers (e.g., 2×, 3×) to approximate the 90th–95th percentile, and ensuring explainability.

Summary of Findings

- **Rule 1 (Botnet Detection):** Two internal devices (`192.168.108.150`, `192.168.108.166`) were flagged for generating more than 2× the mean number of flows and unique destinations.
- **Rule 2 (Data Exfiltration via HTTPS/DNS):** Devices exceeding 3× the average upload on ports 443 or 53 were identified; these cases may warrant immediate inspection.
- **Rule 3 (DNS C&C Detection):** Devices exhibiting excessive DNS query counts ($> 3\times$ mean) or extremely regular query intervals (std $< 0.1\times$ baseline std) were flagged, consistent with periodic beaconing.
- **Rule 4 (New External Destinations):** Internal devices contacting more than 5 previously unseen public IPs in the test period were detected, indicating potential exfiltration to novel endpoints.
- **Rule 5 (External Client Anomalies):** Eight external IPs exhibited average access intervals outside $\pm 2\times$ the standard deviation from the global mean, consistent with scanning or attack patterns.

Overall, eleven internal devices and eight external clients were flagged across all rules. A consolidated report (not shown) lists each flagged IP with the corresponding rule(s) triggered.