HAIDARA Fatimetou, LAHLOU Amal, MARQUES Baptiste, REYNAUD Kenan, SCHULTZ Martin

M2OPSIE - M2SISE

1.1Analyses-des-données-fournies-de-type-IPTABLES

March 7, 2023

[1]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import plotly.express as px
```

[2]:
```
entete=␣
↪["date","IpS","IpD","Protocol","PortS","PortD","Regle","Acces","carteReseau","N
```

[3]:
```
data = pd.read_table('firewall.log', sep=';', names=entete)
data.head()
```

[3]:

	date	IpS	IpD	Protocol	PortS	PortD	\
0	2023-03-06 17:01:20	192.168.43.222	192.168.43.60	TCP	80	4	
1	2023-03-06 17:01:20	192.168.43.222	192.168.43.60	TCP	80	4	
2	2023-03-06 17:01:35	192.168.43.222	192.168.43.60	TCP	80	4	
3	2023-03-06 17:01:35	192.168.43.222	192.168.43.60	TCP	80	4	
4	2023-03-06 17:01:35	192.168.43.222	192.168.43.60	TCP	80	4	

	Regle	Acces	carteReseau	Nan	Inconnu
0	Permit	IN	NaN	NaN	NaN
1	Permit	IN	NaN	NaN	NaN
2	Permit	IN	NaN	NaN	NaN
3	Permit	IN	NaN	NaN	NaN
4	Permit	IN	NaN	NaN	NaN

1 log_fw_3.csv

Nous commençons à importer le fichier le plus simple pour faire nos tests

1.0.1 Import Fichier

[4]:
```
data3= pd.read_csv("log_fw_3.csv",sep=";", header=None)
data3.columns = entete
```

[5]:
```
data3.head()
```

[5]:

	date	IpS	IpD	Protocol	PortS	PortD	Regle	Acces	carteReseau	Nan	Inconnu
0	2023-02-12 03:59:03	66.249.69.180	17.17.17.17	TCP	57630.0	443.0	1.0	PERMIT	eth0	NaN	6.0
1	2023-02-12 03:59:04	66.249.69.178	17.17.17.17	TCP	63808.0	443.0	1.0	PERMIT	eth0	NaN	6.0
2	2023-02-12 03:59:08	221.11.125.141	17.17.17.17	TCP	43123.0	2376.0	999.0	DENY	eth0	NaN	6.0
3	2023-02-12 03:59:18	183.136.225.42	17.17.17.17	TCP	8088.0	70.0	999.0	DENY	eth0	NaN	6.0
4	2023-02-12 03:59:20	38.32.112.34	17.17.17.17	TCP	6356.0	443.0	1.0	PERMIT	eth0	NaN	6.0

1.1 Classement des règles les plus utilisées

[6]:
```
rules3 = data3['Regle']
```

[7]:
```
rule_count3 = {}

# Compter règle
for rule in rules3:
    if rule in rule_count3:
        rule_count3[rule] += 1
    else:
        rule_count3[rule] = 1
```

[8]:
```
# nombre d'occurrences règles
sorted_rules3 = sorted(rule_count3.items(), key=lambda x: x[1], reverse=True)

for rule, count in sorted_rules3:
    print(f"{rule}: {count} occurrences")
```

```
555.0: 542741 occurrences
999.0: 216738 occurrences
1.0: 203912 occurrences
7.0: 12150 occurrences
4.0: 6122 occurrences
8.0: 6116 occurrences
6.0: 5213 occurrences
2.0: 2093 occurrences
9.0: 2093 occurrences
11.0: 1614 occurrences
5.0: 1188 occurrences
777.0: 18 occurrences
nan: 1 occurrences
nan: 1 occurrences
```

la règle la plus utilisé est la règle n°555 suivi de la règle n°999 puis de la règle n°1

1.2 Fournir un histogramme représentant l'utilisation des différents protocoles présents.

1.3 Tcp et Udp ne sont pas forcément nommés ainsi dans le fichier de log,

1.4 une déduction devra être faite selon les ports utilisés, idem pour les protocoles sans nom ou N.A.

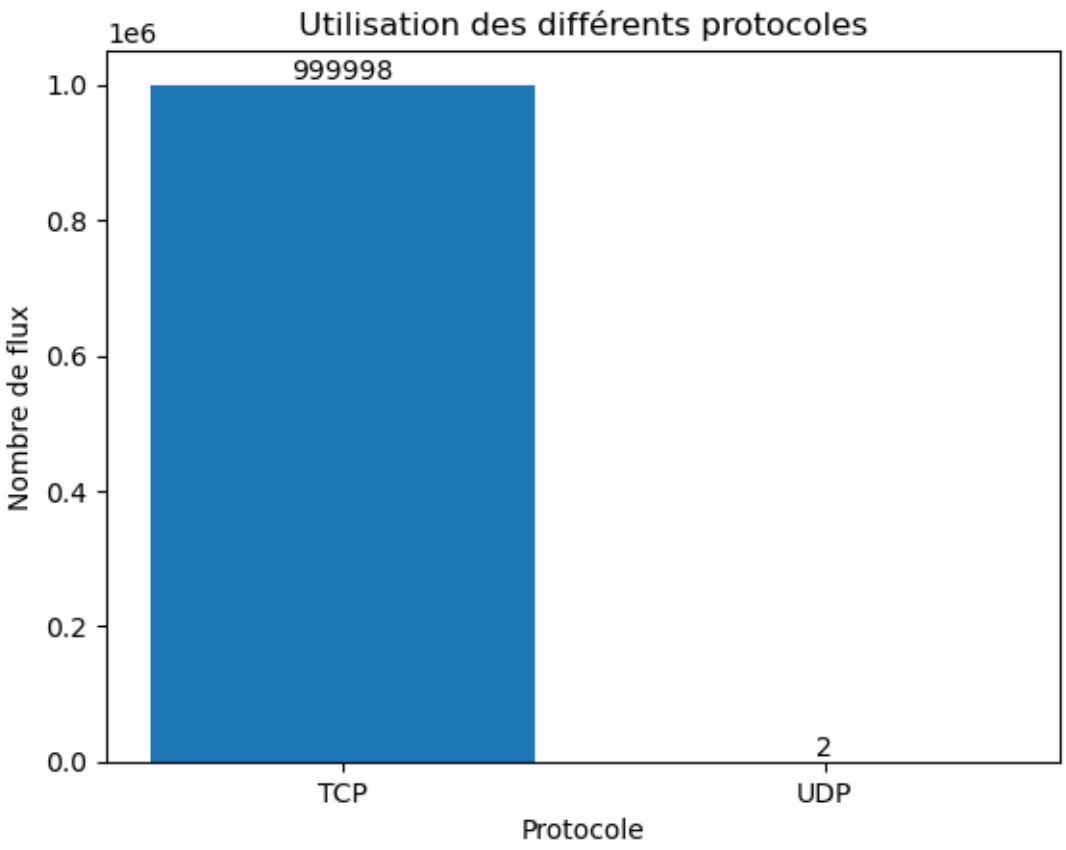
Les champs de la colonne protocole étant vide nous supposons qu'il s'agit des champs du protocole UDP

[9]:
```
protocols = {'TCP': 0, 'UDP': 0, 'N.A.': 0}
```

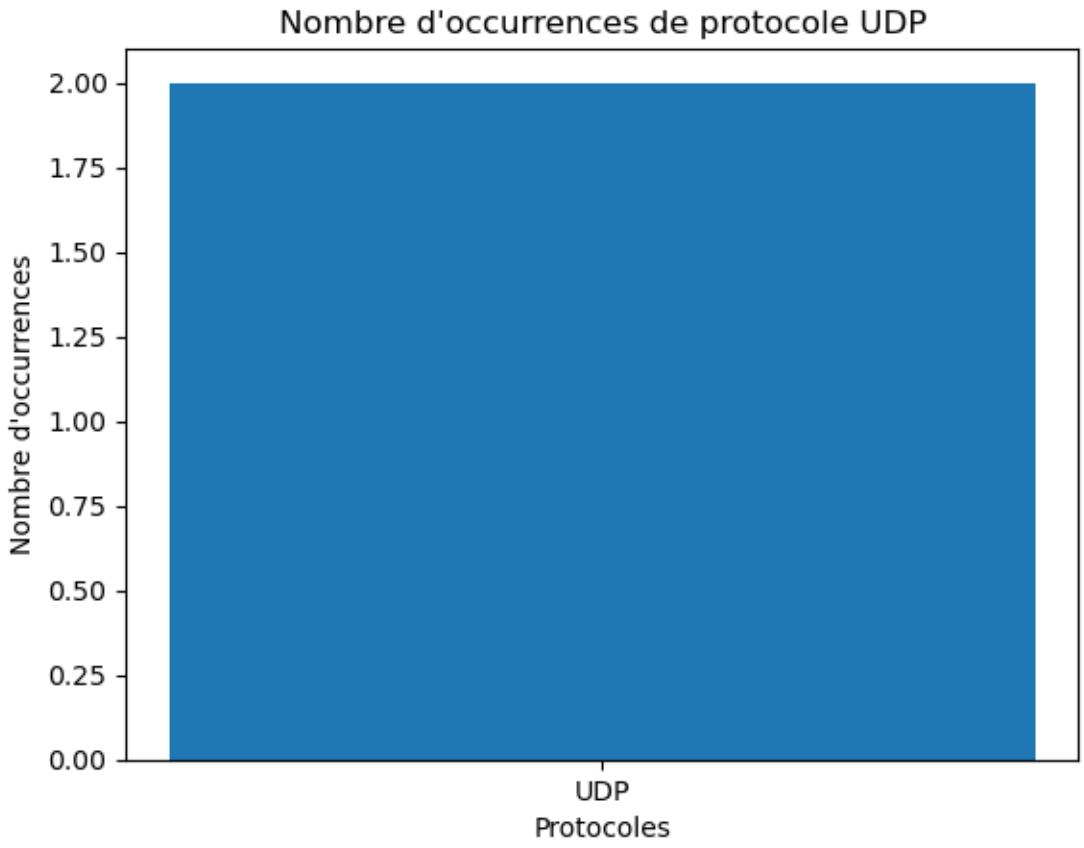
[10]:
```
#compter protocole
for index, row in data3.iterrows():
    protocol = row['Protocol']
    port = row['PortS']
    if protocol == 'N.A.':
        protocols['N.A.'] += 1
    elif protocol == 'TCP' or port in [80, 443, 8080, 8443]:
        protocols['TCP'] += 1
    elif protocol == 'UDP':
        protocols['UDP'] += 1
```

[11]:
```
data3['Protocol'] = data3['Protocol'].fillna('UDP')
```

[12]:
```
# Compter le nombre de flux par protocole
protocol_counts = data3['Protocol'].value_counts()

# Créer un histogramme avec des étiquettes annotées
fig, ax = plt.subplots()
bars = ax.bar(protocol_counts.index, protocol_counts.values)

for i, v in enumerate(protocol_counts.values):
    ax.text(i, v, str(v), ha='center', va='bottom')

# Ajouter des étiquettes d'axe et un titre
ax.set_xlabel('Protocole')
ax.set_ylabel('Nombre de flux')
ax.set_title('Utilisation des différents protocoles')

# Afficher le graphique
plt.show()
```



[13]:
```
# Filtrer les lignes avec le protocole UDP
udp_logs = data3[data3['Protocol'] == 'UDP']

# Compter le nombre de lignes avec le protocole UDP
udp_count = len(udp_logs)

# Générer un graphique montrant le nombre d'occurrences de protocole UDP
plt.bar(['UDP'], [udp_count])
plt.title('Nombre d\'occurrences de protocole UDP')
plt.xlabel('Protocoles')
plt.ylabel('Nombre d\'occurrences')
plt.show()
```



On en déduit que TCP réprésente la quasi-totalité des requêtes de notre serveur

1.5 Top dix des règles les plus utilisées avec le protocole Udp.

[14]:
```
data3.Protocol.unique()
```

[14]:
```
array(['TCP', 'UDP'], dtype=object)
```

[15]:
```
data3.Protocol.value_counts()
```

[15]:
```
TCP    999998
UDP         2
Name: Protocol, dtype: int64
```

[16]:
```
data3.loc[data3['Protocol']=="UDP"]
```

[16]:
```
                      date  IpS  IpD Protocol  PortS  PortD  Regle Acces  \
169957  2023-02-14 12:43:10  NaN  NaN      UDP    NaN    NaN    NaN   NaN
432543  2023-02-19 08:58:27  NaN  NaN      UDP    NaN    NaN    NaN   NaN
```

```
        carteReseau  Nan  Inconnu
169957          NaN  NaN      NaN
432543          NaN  NaN      NaN
```

UDP n'est pas utilisé car chaque en enregistrement qui nous a semblé le concerner est entièrement vide

1.6 Top cinq des règles les plus utilisées avec le protocole Tcp.

[17]:
```
dlog3 = data3
```

[18]:
```
#Graphe des top 5 des régles les plus utilisees avec le protocole TCP
dlog3TCP = dlog3[dlog3['Protocol'] == 'TCP']
portTCP = dlog3TCP['PortD'].value_counts()
#regleTCP = dlog3TCP['Regle'].value_counts()
top5_log3 = portTCP.head(5)
top5_log3
```

[18]:
```
443.0      197150
58428.0     85415
60758.0     85285
52080.0     64677
52691.0     40644
Name: PortD, dtype: int64
```

Le top 5 des ports : le port 443 puis le port 58428 puis le port60758, le port 52080 et enfin le port 52691

[19]:
```
top15_log3 = portTCP.head(15)
top15_log3
```

[19]:
```
443.0      197150
58428.0     85415
60758.0     85285
52080.0     64677
52691.0     40644
56563.0     34902
61762.0     25164
62581.0     20854
58645.0     17374
51327.0     15991
56328.0     12676
22.0        12238
23.0        12150
58642.0     12037
64633.0     10123
Name: PortD, dtype: int64
```

Nous avons réalisé un top 15 pour voir les ports qui ne sont pas présent dans la configuration de base

1.7 Fournir un rapprochement des règles (rule id) par rapport aux ports de destination et les action.

1.8 Cette analyse portera uniquement sur le protocole par défaut Tcp.

1.8.1 Permit

[20]:
```
data3permit = data3.loc[data3['Acces'] == 'PERMIT']
```

[21]:
```
table = pd.crosstab(data3permit.PortD.astype(str), data3permit.Regle)
```

[22]:
```
table.columns
```

[22]:
```
Float64Index([1.0, 2.0, 4.0, 8.0, 9.0, 11.0, 555.0, 777.0], dtype='float64',
name='Regle')
```

[23]:
```
for i in table.columns:
    print(table[i].sort_values(ascending=False).iloc[0:4])
    print('\n')
```

```
PortD
443.0      197150
80.0         6762
61098.0         0
58645.0         0
Name: 1.0, dtype: int64


PortD
3306.0     2093
110.0         0
61098.0       0
58645.0       0
Name: 2.0, dtype: int64


PortD
22.0       6122
110.0         0
61098.0       0
58645.0       0
Name: 4.0, dtype: int64


PortD
22.0       6116
```

```
110.0         0
61098.0       0
58645.0       0
Name: 8.0, dtype: int64


PortD
3306.0     2093
110.0         0
61098.0       0
58645.0       0
Name: 9.0, dtype: int64


PortD
110.0      1614
61098.0       0
58645.0       0
58755.0       0
Name: 11.0, dtype: int64


PortD
58428.0    85415
60758.0    85282
52080.0    64677
52691.0    40641
Name: 555.0, dtype: int64


PortD
2224.0     18
110.0       0
61098.0     0
58645.0     0
Name: 777.0, dtype: int64
```

[24]:
```
for i in table.columns:

    px.histogram(x=table[i].sort_values(ascending=False).iloc[0:4],
                 y= pd.DataFrame(table[i].sort_values(ascending=False).iloc[0:
 ↪4]).index,
                 nbins=5).show()
    print('\n')
```

Pour chaque actions Acceptées du protocol TCP nous avons réalisé un top 5 des ports les plus solicités par règle.

1.8.2 Deny

[25]:
```
data3deny = data3.loc[data3['Acces'] == 'DENY']
```

[26]:
```
tableD = pd.crosstab(data3deny.PortD.astype(str), data3deny.Regle)
```

[27]:
```
for i in tableD.columns:
    print(tableD[i].sort_values(ascending=False).iloc[0:4])
    print('\n')
```

```
PortD
21.0       1012
20.0        176
1.0           0
51494.0       0
Name: 5.0, dtype: int64


PortD
445.0     4220
3389.0     902
135.0       91
1.0          0
Name: 6.0, dtype: int64
```

```
PortD
23.0       12150
1.0            0
51490.0        0
51480.0        0
Name: 7.0, dtype: int64


PortD
5555.0    1307
8443.0    1238
8080.0    1126
1433.0     981
Name: 999.0, dtype: int64
```

[28]:
```
for i in tableD.columns:

    px.histogram(x=tableD[i].sort_values(ascending=False).iloc[0:4],
                 y=pd.DataFrame(tableD[i].sort_values(ascending=False).iloc[0:
↪4]).index,
                 nbins=5).show()
    print('\n')
```

Pour chaque actions Refusées du protocol TCP nous avons réalisé un top 5 des ports les plus solicités par règle.

2 log_fw_4.csv

Ensuite nous fesons un nos analyses sur le gros fichiers

2.0.1 Import Fichier

[29]:
```
data4 = pd.read_csv("log_fw_4.csv",sep=";", header=None)
data4.columns = entete
```

[30]:
```
data4.head()
```

[30]:

	date	IpS	IpD	Protocol	PortS	PortD	Regle	Acces	carteReseau	Nan	Inconnu
0	2022-02-23 09:30:55	89.248.165.121	17.17.17.17	TCP	50242.0	25918.0	999.0	DENY	eth0	NaN	6.0
1	2022-02-23 09:30:57	159.223.115.197	17.17.17.17	TCP	47162.0	24474.0	999.0	DENY	eth0	NaN	6.0
2	2022-02-23 09:31:03	89.248.165.121	17.17.17.17	TCP	50242.0	23937.0	999.0	DENY	eth0	NaN	6.0
3	2022-02-23 09:31:09	91.240.118.73	17.17.17.17	TCP	41839.0	1996.0	999.0	DENY	eth0	NaN	6.0
4	2022-02-23 09:31:12	51.222.253.20	17.17.17.17	TCP	46712.0	443.0	1.0	PERMIT	eth0	NaN	6.0

2.1 Classement des règles les plus utilisées

[31]:
```
rules = data4['Regle']
```

[32]:
```
rule_count = {}

# Compter règle
for rule in rules:
    if rule in rule_count:
        rule_count[rule] += 1
    else:
        rule_count[rule] = 1
```

[33]:
```
# nombre d'occurrences règles
sorted_rules = sorted(rule_count.items(), key=lambda x: x[1], reverse=True)

for rule, count in sorted_rules:
    print(f"{rule}: {count} occurrences")
```

```
555.0: 9387198 occurrences
1.0: 5092705 occurrences
999.0: 2373712 occurrences
7.0: 207395 occurrences
4.0: 164194 occurrences
8.0: 164111 occurrences
6.0: 104076 occurrences
2.0: 61784 occurrences
9.0: 61784 occurrences
11.0: 35993 occurrences
5.0: 20668 occurrences
777.0: 170 occurrences
nan: 1 occurrences
```

```
nan: 1 occurrences
nan: 1 occurrences
nan: 1 occurrences
nan: 1 occurrences
nan: 1 occurrences
nan: 1 occurrences
nan: 1 occurrences
nan: 1 occurrences
nan: 1 occurrences
nan: 1 occurrences
nan: 1 occurrences
nan: 1 occurrences
```

la règle la plus utilisé est la règle n°555 suivi de la règle n°1 puis de la règle n°999. Nous remarqons une différence avec le premier fichier, les positions de la règle n°1 et n°999 étaient inversées

2.2 Fournir un histogramme représentant l'utilisation des différents protocoles présents.

2.3 Tcp et Udp ne sont pas forcément nommés ainsi dans le fichier de log,

2.4 une déduction devra être faite selon les ports utilisés, idem pour les protocoles sans nom ou N.A.

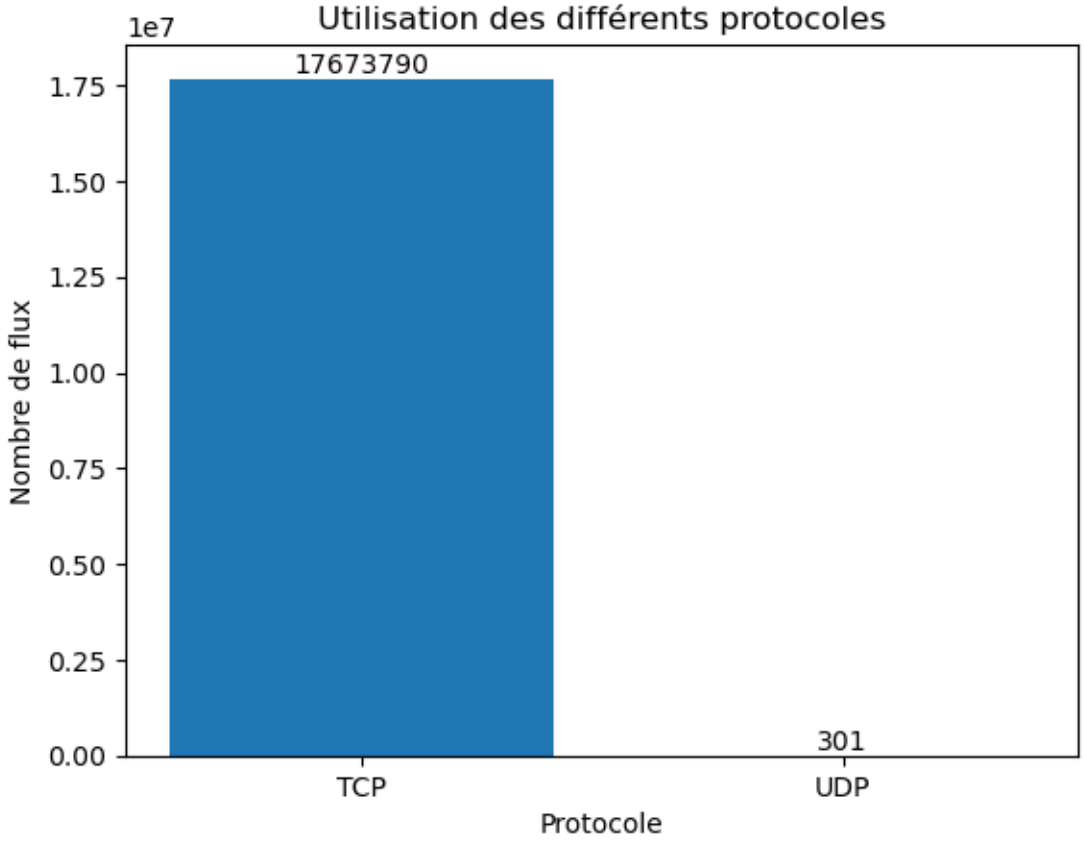
[34]:
```
protocols = {'TCP': 0, 'UDP': 0, 'N.A.': 0}
```

[35]:
```
#compter protocole
for index, row in data4.iterrows():
    protocol = row['Protocol']
    port = row['PortS']
    if protocol == 'N.A.':
        protocols['N.A.'] += 1
    elif protocol == 'TCP' or port in [80, 443, 8080, 8443]:
        protocols['TCP'] += 1
    elif protocol == 'UDP':
        protocols['UDP'] += 1
```

[36]:
```
data4['Protocol'] = data4['Protocol'].fillna('UDP')
```

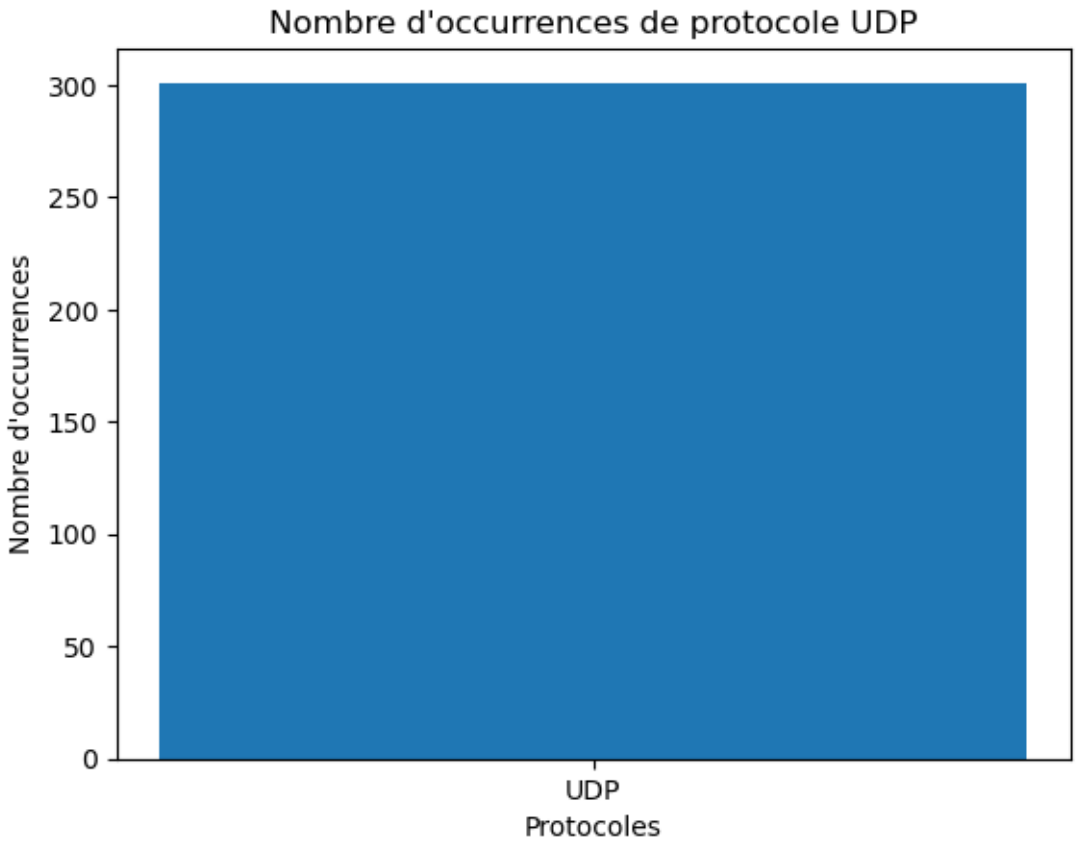
[37]:
```
# Compter le nombre de flux par protocole
protocol_counts4 = data4['Protocol'].value_counts()

# Créer un histogramme avec des étiquettes annotées
fig, ax = plt.subplots()
bars = ax.bar(protocol_counts.index, protocol_counts4.values)

for i, v in enumerate(protocol_counts4.values):
    ax.text(i, v, str(v), ha='center', va='bottom')
```

```
# Ajouter des étiquettes d'axe et un titre
ax.set_xlabel('Protocole')
ax.set_ylabel('Nombre de flux')
ax.set_title('Utilisation des différents protocoles')

# Afficher le graphique
plt.show()
```



[38]:
```
# Filtrer les lignes avec le protocole UDP
udp_logs = data4[data4['Protocol'] == 'UDP']

# Compter le nombre de lignes avec le protocole UDP
udp_count = len(udp_logs)

# Générer un graphique montrant le nombre d'occurrences de protocole UDP
plt.bar(['UDP'], [udp_count])
plt.title('Nombre d\'occurrences de protocole UDP')
plt.xlabel('Protocoles')
plt.ylabel('Nombre d\'occurrences')
```

```
plt.show()
```



Même conclusion ici le TCP représente la majorité des reqêtes

2.5 Top dix des règles les plus utilisées avec le protocole Udp.

[39]:
```
data4.Protocol.unique()
```

[39]:
```
array(['TCP', 'UDP'], dtype=object)
```

[40]:
```
data4.Protocol.value_counts()
```

[40]:
```
TCP    17673790
UDP          301
Name: Protocol, dtype: int64
```

[41]:
```
#Top dix des règles les plus utilisées avec le protocole Udp.
pd.set_option('display.max_rows', 300)
data4.loc[data4['Protocol']=='UDP']
```

[41]:
```
                         date  IpS  IpD Protocol  PortS  PortD  Regle Acces  \
10579     2022-02-24 00:19:04  NaN  NaN      UDP    NaN    NaN    NaN   NaN
80673     2022-02-25 01:10:58  NaN  NaN      UDP    NaN    NaN    NaN   NaN
95883     2022-02-26 00:11:35  NaN  NaN      UDP    NaN    NaN    NaN   NaN
153715    2022-02-26 23:35:20  NaN  NaN      UDP    NaN    NaN    NaN   NaN
176093    2022-02-27 23:47:02  NaN  NaN      UDP    NaN    NaN    NaN   NaN
…                         …    …   …        …      …      …      …    …
15241262  2023-01-15 23:20:56  NaN  NaN      UDP    NaN    NaN    NaN   NaN
15267138  2023-01-17 00:49:01  NaN  NaN      UDP    NaN    NaN    NaN   NaN
15326847  2023-01-18 00:06:59  NaN  NaN      UDP    NaN    NaN    NaN   NaN
16844048  2023-02-14 12:43:10  NaN  NaN      UDP    NaN    NaN    NaN   NaN
17106634  2023-02-19 08:58:27  NaN  NaN      UDP    NaN    NaN    NaN   NaN

          carteReseau  Nan  Inconnu
10579             NaN  NaN      NaN
80673             NaN  NaN      NaN
95883             NaN  NaN      NaN
153715            NaN  NaN      NaN
176093            NaN  NaN      NaN
…                 …    …        …
15241262          NaN  NaN      NaN
15267138          NaN  NaN      NaN
15326847          NaN  NaN      NaN
16844048          NaN  NaN      NaN
17106634          NaN  NaN      NaN

[301 rows x 11 columns]
```

Même anomalie dans ce fichier ci les enregistrement considérés comme UDP sont tous vides

2.6 Top cinq des règles les plus utilisées avec le protocole Tcp.

[42]:
```
dlog4 = data4
```

[43]:
```
#Top 5 des règles les plus utilisées avec le protocole TCP
#Graphe des top 5 des régles les plus utilisees avec le protocole TCP
dlog4TCP = dlog4[dlog4['Protocol'] == 'TCP']
portTCP = dlog4TCP['PortD'].value_counts()
#regleTCP = dlog4TCP['Regle'].value_counts()
top5_log4 = portTCP.head(5)
top5_log4
```

[43]:
```
443.0      4371970
80.0        720735
22.0        328305
23.0        207395
64786.0     135468
Name: PortD, dtype: int64
```

Le top 5 des ports : le port 443 puis le port 80 puis le port 22, le port 23 et enfin le port 64786

Nous avons réalisé un top 15 pour voir les ports qui ne sont pas présent dans la configuration de base du Firewall

[44]:
```
top15_log4 = portTCP.head(15)
top15_log4
```

[44]:
```
443.0      4371970
80.0        720735
22.0        328305
23.0        207395
64786.0     135468
3306.0      123568
62948.0     100480
51623.0      97140
58428.0      94484
445.0        88512
60758.0      85294
50233.0      82096
59678.0      81541
62829.0      81022
63883.0      80734
Name: PortD, dtype: int64
```

2.7 Fournir un rapprochement des règles (rule id) par rapport aux ports de destination et les action.

2.8 Cette analyse portera uniquement sur le protocole par défaut Tcp.

2.8.1 Permit

[45]:
```
data4permit = data4.loc[data4['Acces'] == 'PERMIT']
table4 = pd.crosstab(data4permit.PortD.astype(str), data4permit.Regle)
table4.columns
```

[45]:
```
Float64Index([1.0, 2.0, 4.0, 8.0, 9.0, 11.0, 555.0, 777.0], dtype='float64',
name='Regle')
```

[46]:
```
for i in table4.columns:
    print(table4[i].sort_values(ascending=False).iloc[0:4])
    print('\n')
```

```
PortD
443.0      4371970
80.0        720735
60245.0          0
59792.0          0
Name: 1.0, dtype: int64
```

```
PortD
3306.0     61784
110.0          0
60245.0        0
59792.0        0
Name: 2.0, dtype: int64


PortD
22.0       164194
110.0           0
60245.0         0
59792.0         0
Name: 4.0, dtype: int64


PortD
22.0       164111
110.0           0
60245.0         0
59792.0         0
Name: 8.0, dtype: int64


PortD
3306.0     61784
110.0          0
60245.0        0
59792.0        0
Name: 9.0, dtype: int64


PortD
110.0      35993
60537.0        0
59792.0        0
59843.0        0
Name: 11.0, dtype: int64


PortD
64786.0    135459
62948.0    100468
51623.0     97128
58428.0     94475
Name: 555.0, dtype: int64
```

```
PortD
2087.0     84
2224.0     82
7.0         4
60245.0     0
Name: 777.0, dtype: int64
```

[47]:
```
for i in table4.columns:

    px.histogram(x=table4[i].sort_values(ascending=False).iloc[0:4],
                 y= pd.DataFrame(table4[i].sort_values(ascending=False).iloc[0:
 ↪4]).index,
                 nbins=5).show()
    print('\n')
```

Pour chaque les actions Acceptées du protocol TCP nous avons réalisé un top 5 des ports les plus solicités par règle. Ici dans le fichier log_fw_4.csv

2.8.2 Deny

[48]:
```
data4deny = data4.loc[data4['Acces'] == 'DENY']
table4D = pd.crosstab(data4deny.PortD.astype(str), data4deny.Regle)
```

[49]:
```
for i in table4D.columns:
    print(table4D[i].sort_values(ascending=False).iloc[0:4])
```

```
    print('\n')
```

```
PortD
21.0       18188
20.0        2480
49326.0        0
49314.0        0
Name: 5.0, dtype: int64


PortD
445.0      88512
3389.0     14163
135.0       1401
49326.0        0
Name: 6.0, dtype: int64


PortD
23.0       207395
1.0             0
49312.0         0
49314.0         0
Name: 7.0, dtype: int64


PortD
8080.0    48278
8443.0    30595
5555.0    24653
1433.0    19754
Name: 999.0, dtype: int64
```

[50]:
```
for i in table4D.columns:

    px.histogram(x=table4D[i].sort_values(ascending=False).iloc[0:4],
                 y=pd.DataFrame(table4D[i].sort_values(ascending=False).iloc[0:
 ↪4]).index,
                 nbins=5).show()
    print('\n')
```

Pour chaque actions Refusées du protocol TCP nous avons réalisé un top 5 des ports les plus solicités par règle.

[ ]:

Rapportchallenge_sise_opsi

```
library(dplyr)
```

```
##
## Attachement du package : 'dplyr'
```

```
## Les objets suivants sont masqués depuis 'package:stats':
##
##     filter, lag
```

```
## Les objets suivants sont masqués depuis 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
library(scales)
library(lubridate)
```

```
##
## Attachement du package : 'lubridate'
```

```
## Les objets suivants sont masqués depuis 'package:base':
##
##     date, intersect, setdiff, union
```

```
dlog3 <- read.table("log_fw_3.csv",sep=";",header=FALSE)
```

```
dlog4 <- read.table("log_fw_4.csv",sep=";",header=FALSE)
```

```
colnames(dlog3) <- c("date","IpS","IpD","Protocol","PortS","PortD","Regle","Acces","carteReseau","Nan",
```

```
colnames(dlog4) <- c("date","IpS","IpD","Protocol","PortS","PortD","Regle","Acces","carteReseau","Nan",
```

```
dlog3TCP <- dlog3[which(dlog3$Protocol == "TCP"), ]
portTCP <- dlog3TCP %>%count(PortD)
top5_log3 <- head(portTCP[order(-portTCP$n),],5)
top5_log3
```

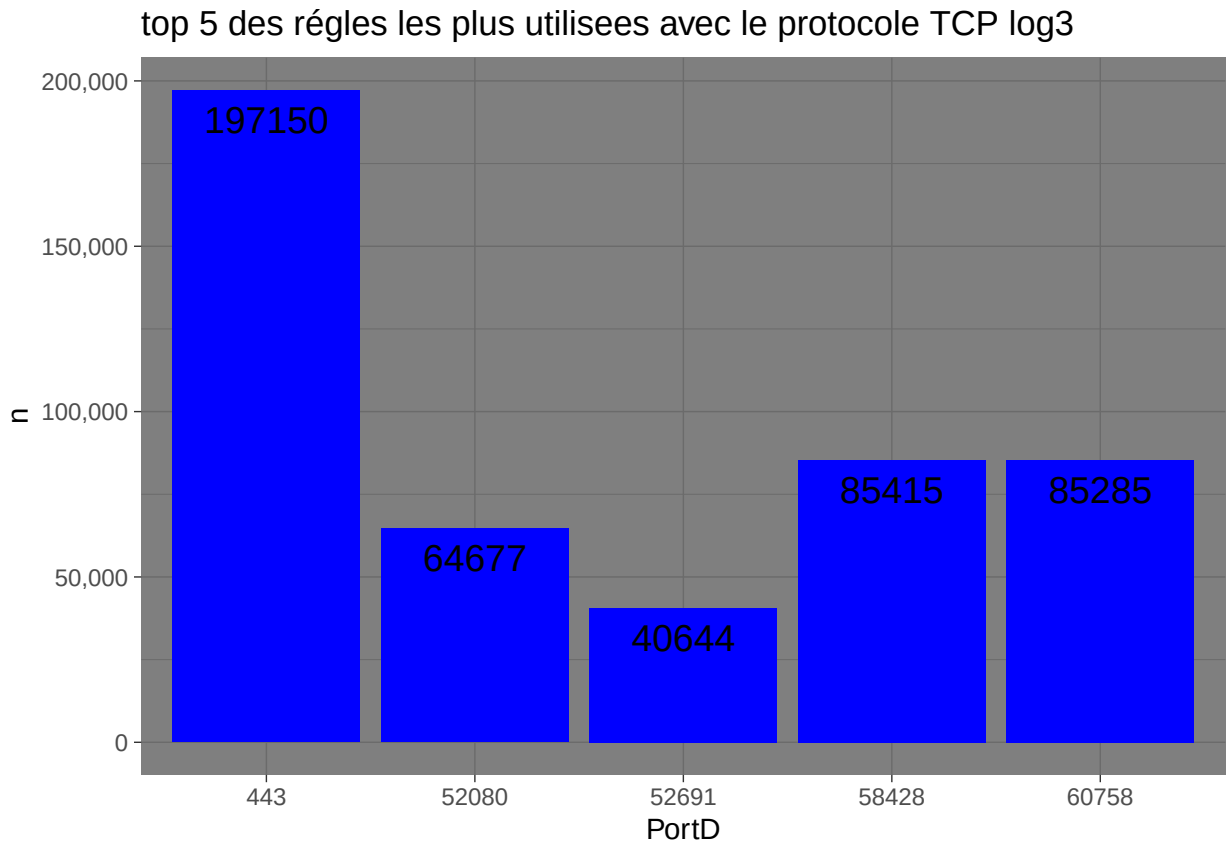
```
##       PortD      n
## 402     443 197150
## 48373 58428  85415
## 50560 60758  85285
## 42690 52080  64677
## 43226 52691  40644
```

```
dlog4TCP <- dlog4[which(dlog4$Protocol == "TCP"), ]
portTCP <- dlog4TCP %>%count(PortD)
top5_log4 <- head(portTCP[order(-portTCP$n),],5)
top5_log4
```

```
##       PortD       n
## 442     443 4371970
## 79       80  720735
## 22       22  328305
## 23       23  207395
## 64785 64786  135468
```

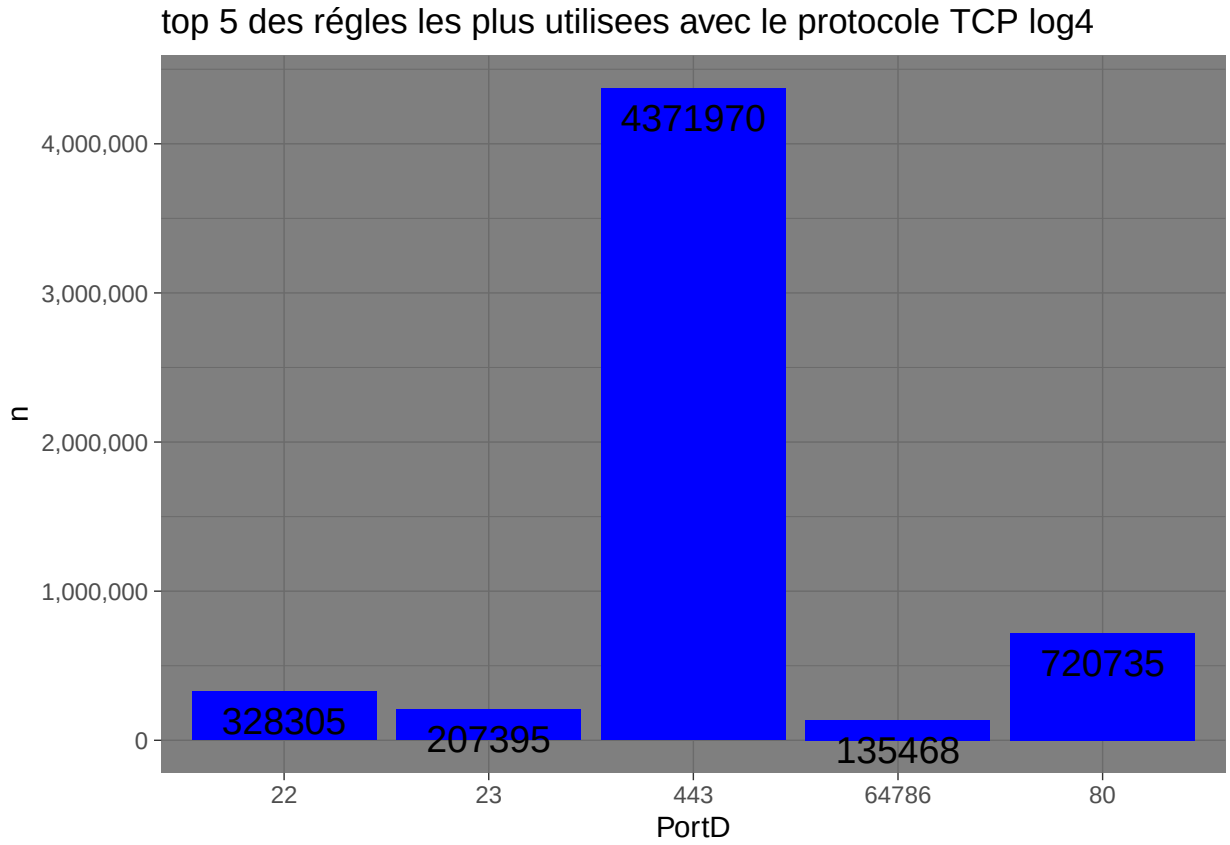
Graphe des top 5 des règles les plus utilisées avec le protocole TCP log3

```
top5_log3$PortD <- as.character(top5_log3$PortD)
ggplot(data=top5_log3, aes(x=PortD, y=n)) +
geom_bar(stat="identity",fill="blue" )+
  ggtitle("top 5 des régles les plus utilisees avec le protocole TCP log3")+
theme_dark()+
scale_y_continuous(labels = comma)+
geom_text(aes(label=n), vjust=1.6, colour="black", size =5)
```



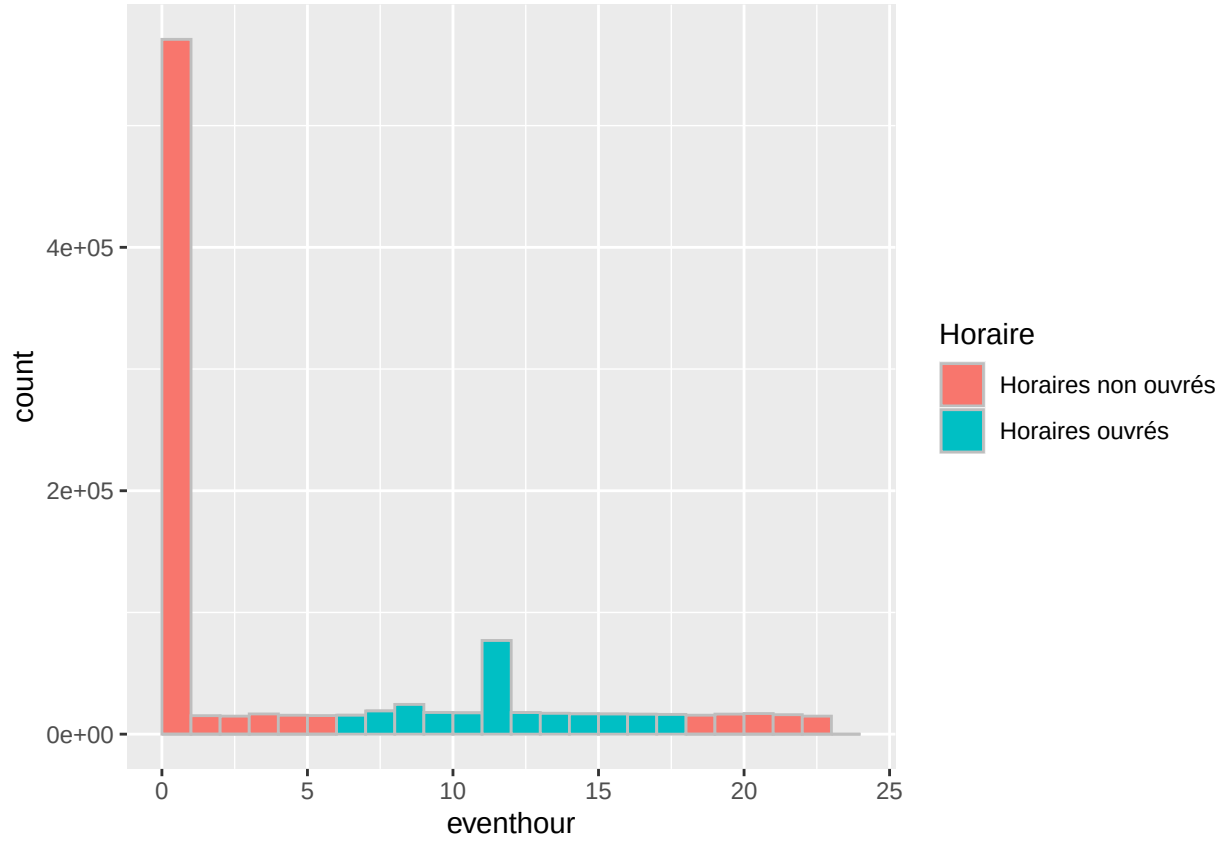
Graphe des top 5 des règles les plus utilisees avec le protocole TCP log4

```
top5_log4$PortD <- as.character(top5_log4$PortD)
ggplot(data=top5_log4, aes(x=PortD, y=n)) +
geom_bar(stat="identity",fill="blue" )+
  ggtitle("top 5 des régles les plus utilisees avec le protocole TCP log4")+
theme_dark()+
scale_y_continuous(labels = comma)+
geom_text(aes(label=n), vjust=1.6, colour="black", size =5)
```



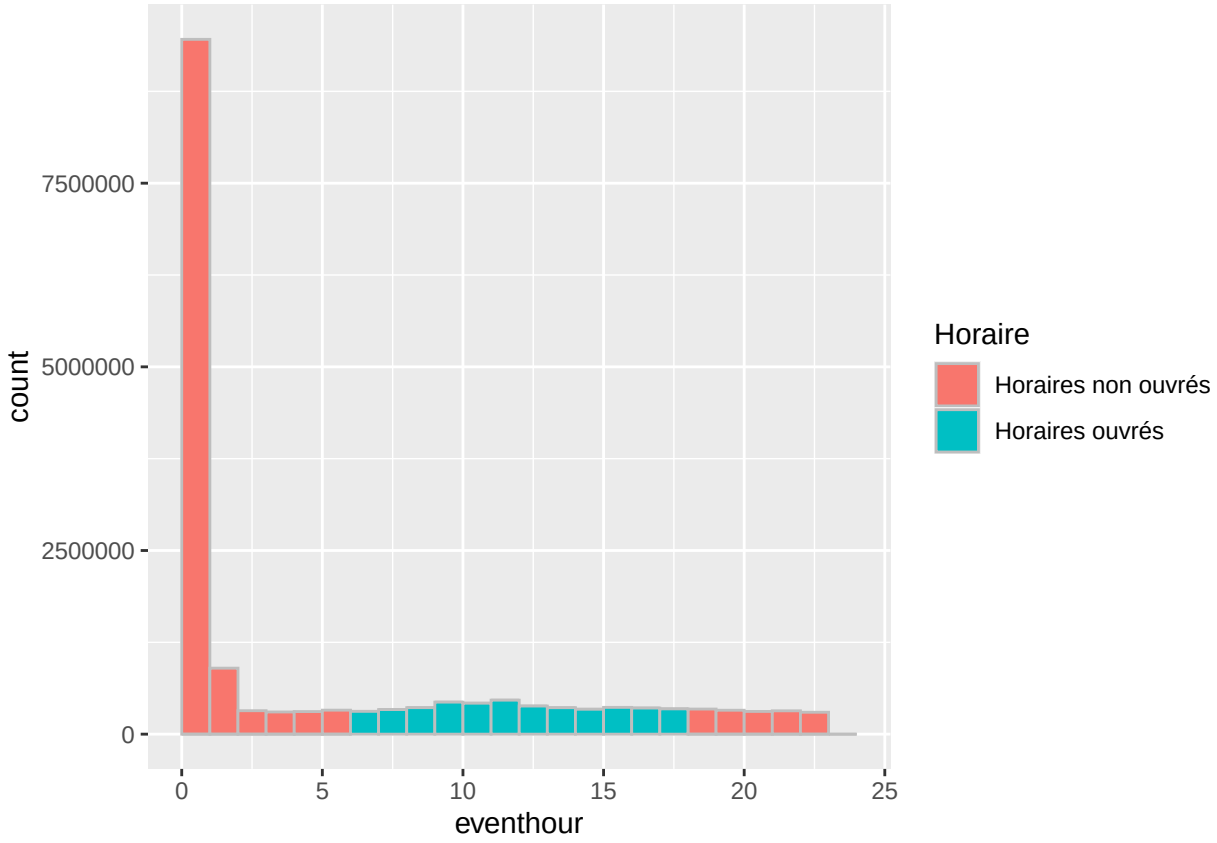
Visualisation des heures des requêtes pour le ficher log_fw_3

```
hour_of_event <- hour(dlog3$date)
dlog3 <- data.frame(datetime=dlog3$date,eventhour=hour_of_event)
dlog3$Horaire <- dlog3$eventhour %in% seq(7,18)
dlog3$Horaire[dlog3$Horaire=='TRUE']<-"Horaires ouvrés"
dlog3$Horaire[dlog3$Horaire=='FALSE']<-"Horaires non ouvrés"
ggplot(dlog3,aes(x=eventhour,fill=Horaire)) + geom_histogram(breaks=seq(0,24),colour="grey")
```



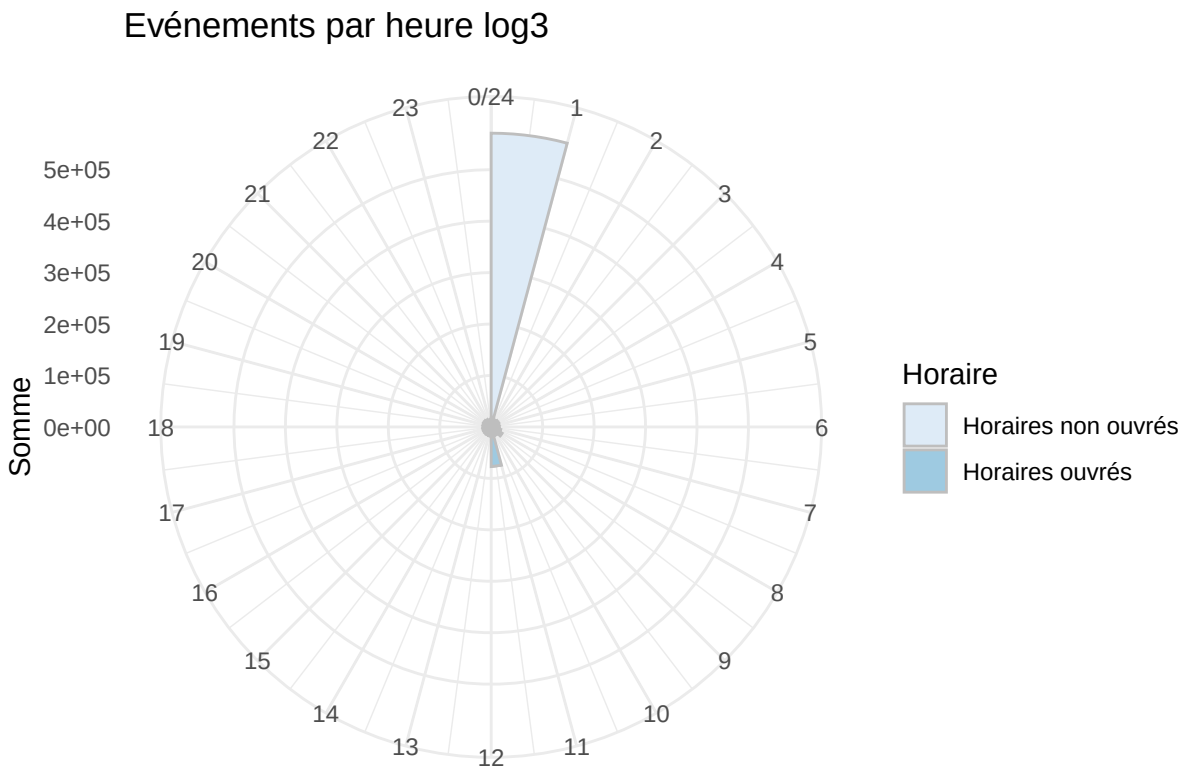
Visualisation des heures de requêtes pour le ficher log_fw_4

```
hour_of_event <- hour(dlog4$date)
dlog4 <- data.frame(datetime=dlog4$date,eventhour=hour_of_event)
dlog4$Horaire <- dlog4$eventhour %in% seq(7,18)
dlog4$Horaire[dlog4$Horaire=='TRUE']<-"Horaires ouvrés"
dlog4$Horaire[dlog4$Horaire=='FALSE']<-"Horaires non ouvrés"
ggplot(dlog4,aes(x=eventhour,fill=Horaire)) + geom_histogram(breaks=seq(0,24),colour="grey")
```

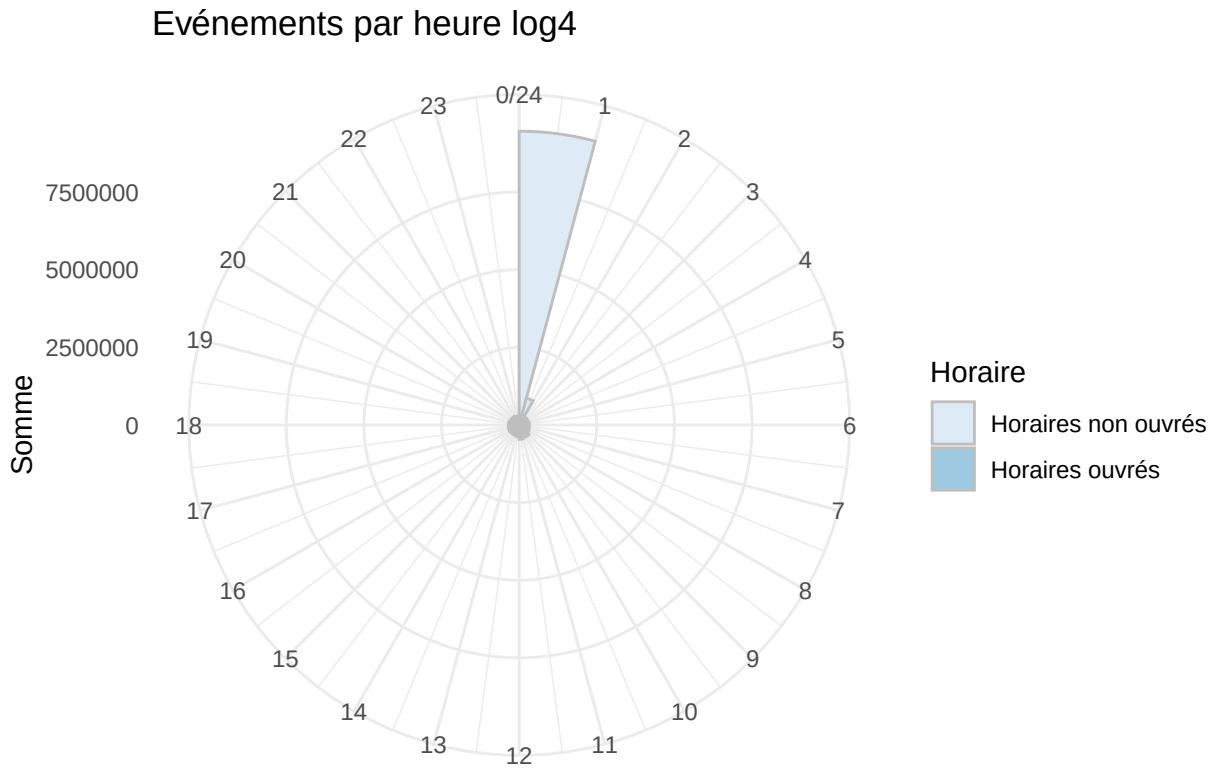


Une vision plus simple

```
hour_of_event <- hour(dlog3$date)
dlog3 <- data.frame(datetime=dlog3$date,eventhour=hour_of_event)
dlog3$Horaire <- dlog3$eventhour %in% seq(7,18)
dlog3$Horaire[dlog3$Horaire=='TRUE']<-"Horaires ouvrés"
dlog3$Horaire[dlog3$Horaire=='FALSE']<-"Horaires non ouvrés"
ggplot(dlog3,aes(x=eventhour,fill=Horaire)) + geom_histogram(breaks=seq(0,24),colour="grey")+
coord_polar(start=0)+theme_minimal()+scale_fill_brewer()+ylab("Somme")+ggtitle("Evénements par heure lo
scale_x_continuous("",limits =c(0,24),breaks=seq(0,24),labels=seq(0,24))
```




```
hour_of_event <- hour(dlog4$date)
dlog4 <- data.frame(datetime=dlog4$date,eventhour=hour_of_event)
dlog4$Horaire <- dlog4$eventhour %in% seq(7,18)
dlog4$Horaire[dlog4$Horaire=='TRUE']<-"Horaires ouvrés"
dlog4$Horaire[dlog4$Horaire=='FALSE']<-"Horaires non ouvrés"
ggplot(dlog4,aes(x=eventhour,fill=Horaire)) + geom_histogram(breaks=seq(0,24),colour="grey")+
coord_polar(start=0)+theme_minimal()+scale_fill_brewer()+ylab("Somme")+ggtitle("Evénements par heure log
scale_x_continuous("",limits =c(0,24),breaks=seq(0,24),labels=seq(0,24))
```


Evénements par heure log4
Somme
7500000
5000000
2500000
0
0/24
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
Horaire
Horaires non ouvrés
Horaires ouvrés