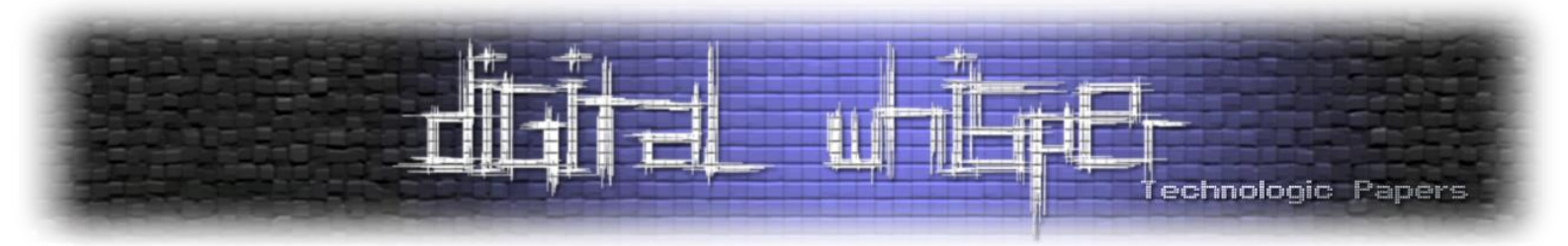


פתרון אתגרי AI, Data Science - אתגר השב"כ 2021

מאת Dvd848, zVaz ו-YaakovCohen88

הקדמה

במהלך חודש ינואר 2021 עלה לאוויר למשך כשבועיים אתגר הגיוס השנתי של השב"כ. האתגר עסק במגוון קטגוריות (Pwn, Reversing, Web, AI, Signal Processing, Data Science). את פתרון אתגרי ה-Web אפשר למצוא ב[מאמר של עידו פרנקנטל](#) מגליון 126, את פתרון אתגרי ה-Pwn והרברסינג אפשר למצוא ב[מאמר של דן אלעזרי](#) מגליון 127, ואנחנו נציג בגליון הזה את הפתרונות שלנו לאתגרי ה-AI וה-Data Sceince (מלבד אחד אם חייבים לדייק). אז מי מרים את הכפפה ומפרסם בגליון הבא את פתרון אתגרי ה-Signal Processing?

אתגרי ה-Data Science

Welcome to the Data Science for Cyber Defense Challenge!

In this challenge you are presented with network logs of a mid-size organization. An anonymous source reported that the network has been compromised by not just one, but 4 different types of malware. Your objective is to analyze the logs and discover the infected endpoints. Each correct identification of endpoints that have been attacked by a given malware type will grant you a flag.

All together there are 4 flags, corresponding to the 4 malware types. The network logs are given in the challenge.csv file which contains the following columns: **timestamp, src_ip, dst_ip, src_port, dst_port, protocol, payload**

Submission Guidelines

Submission of suspected endpoints should be in the example submission_example.csv format below. Each submission should contain the endpoints infected by a single malware type. Note that a single endpoint may be infected by at most one malware type.

Scoring

The submission score is the maximal corresponding to any of the four malware types. A minimal score of 0.8 is required in order to gain the flag which corresponds to the best-detected malware type.

Can you gain all four flags?

לאתגר צורף קובץ לוג ענק (24456694 שורות).

פתרון:

כאמור, עלינו לנתח את התעבורה בקובץ ולאתר את העקבות של ארבע נוזקות שונות שמסתתרות בתוך הרשת הארגונית.

נתחיל עם טעימה מקובץ הלוג:

```
root@kali:/media/sf_CTFs/shabak/DataScience# cat challenge.csv | head
timestamp,src_ip,dst_ip,src_port,dst_port,protocol,payload
2020-06-21 00:00:02.892702,120.18.164.170,120.18.53.84,2459,53,UDP,218
2020-06-21 00:00:03.771702,120.18.53.84,120.18.164.170,53,2459,UDP,285
2020-06-21 00:00:03.989702,120.18.164.170,148.26.83.117,2495,443,TCP,142
2020-06-21 00:00:04.547702,148.26.83.117,120.18.164.170,443,2495,TCP,130
2020-06-21 00:00:05.170953,120.18.187.161,120.18.63.42,3487,1993,TCP,69
2020-06-21 00:00:05.691953,120.18.63.42,120.18.187.161,1993,3487,TCP,243
2020-06-21 00:00:06.124702,120.18.164.170,148.26.83.117,2495,443,TCP,110
2020-06-21 00:00:07.230702,148.26.83.117,120.18.164.170,443,2495,TCP,277
2020-06-21 00:00:08.522702,120.18.164.170,148.26.83.117,2495,443,TCP,198
```

כפי שניתן לראות, כל רשומה כוללת תאריך ושעה, כתובת IP ופורט עבור המקור והיעד, סוג הפרוטוקול וכמות הבתים שנשלחו.

באמצעות מידע זה בלבד עלינו למצוא אנומליות שעשויות להעיד על פעילות זדונית. למשל, אולי נחפש שרת שפונה לכמות חריגה של שרתים אחרים (כדי להדביק אותם בנוזקה או לנהל אותם לאחר שנדבקו), או שרת ששולח מחוץ לארגון כמות חריגה של נתונים (כדי להדליף החוצה את סודות הארגון).

לצורך ניתוח הנתונים אנחנו נשתמש ב-pandas - ספריית פייתון שתוכננה בדיוק עבור משימות אנליזה מעין אלו. במהלך האתגר מצאנו רפרנס נהדר עבור סוג כזה של אתגרים ונעזרנו בו רבות.

נתחיל מהכנת השטח לשאילתות שיגיעו לאחר מכן באמצעות הגדרת מספר משתני עזר:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import gc

df = pd.read_csv(
    'challenge.csv',
    header = 0,
    names= ["timestamp", "src_ip", "dst_ip", "src_port", "dst_port",
"protocol", "payload"]
)

df['timestamp'] = pd.to_datetime(df.timestamp, format='%Y-%m-%d
%H:%M:%S.%f')
df['hour']      = df.timestamp.dt.hour.astype('uint8')
df['minute']    = df.timestamp.dt.minute.astype('uint8')
df['src_port']  = df['src_port'].astype('uint16')
df['dst_port']  = df['dst_port'].astype('uint16')
df.head()

all_ips = set(df['src_ip'].unique()) | set(df['dst_ip'].unique())
print('Unique src:', df['src_ip'].nunique())
```

```
print('Unique dst:', df['dst_ip'].nunique())
print('Total Unique IPs:', len(all_ips))

ip_type = pd.CategoricalDtype(categories=all_ips)
df['src_ip'] = df['src_ip'].astype(ip_type)
df['dst_ip'] = df['dst_ip'].astype(ip_type)
gc.collect()
```

פה אנחנו בסך הכל קוראים את הקובץ לתוך מבנה נתונים של pandas, כאשר המידע עצמו מומר לטיפוס המתאים ביותר לשם כך (למשל - התאריך מומר לאובייקט תאריך). אנחנו גם מייצרים קבוצה של כלל כתובות ה-IP (כתובות מקור + כתובות יעד) ואפשרות לגשת לכתובות מקור ויעד בצורה נוחה. מהרצת הקוד אנחנו רואים שישנן 5497 כתובות IP שונות:

```
Unique src: 5497
Unique dst: 5497
Total Unique IPs: 5497
```

נרצה כעת להבין מהו טווח הכתובות של הארגון שלנו. לשם כך, נחפש את התחיליות שחוזרות על עצמן הכי הרבה:

```
def get_network_prefix(s):
    return s.str.split(".").str[0]
df['src_network_prefix'] = get_network_prefix(df['src_ip'])
df.drop_duplicates("src_ip").groupby("src_network_prefix").size().sort_v
alues(ascending=False)
```

נריץ ונקבל:

```
src_network_prefix
120    523
142     38
131     32
100     30
7       30
      ...
170     11
213     11
8       10
193     10
95       9
Length: 254, dtype: int64
```

ניתן לראות פה ש-120 הינה התחילית הנפוצה ביותר, בפער ברור.

נניח שכתובות שמתחילות ב-120 שייכות לארגון שלנו, ולשם הנוחות, נייצר מספר משתני עזר נוספים:

```
def is_internal(s):
    return s.str.startswith('120.')

df['src_int'] = is_internal(df['src_ip'])
df['dst_int'] = is_internal(df['dst_ip'])
```

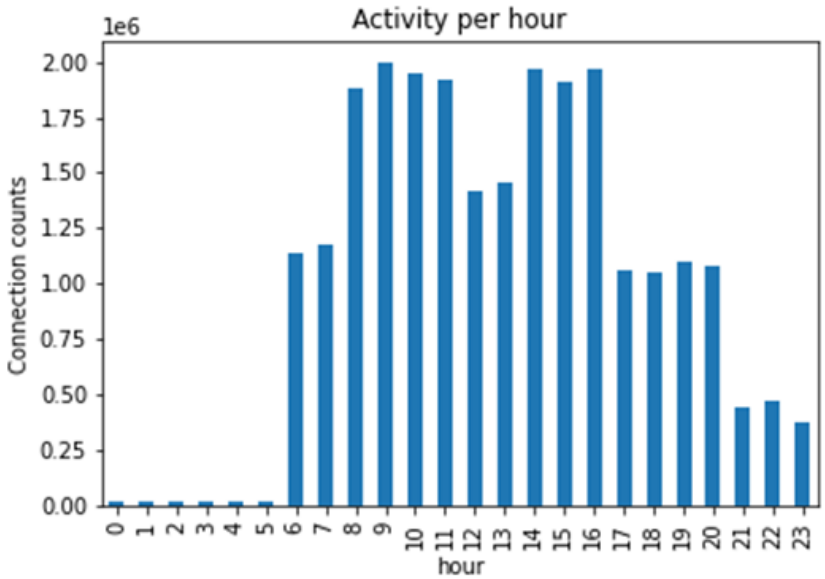
כך נוכל לגשת בקלות לכתובות מהארגון שמשמשות ככתובת מקור או ככתובת יעד. כעת, נתחיל לחפש אנומליות, נוזקות ודגלים.

דגל ראשון (150 נקודות)

נתחיל מניתוח שעות העבודה בארגון ונחפש תעבורה חריגה מחוץ לשעות העבודה. את שעות העבודה אפשר למצוא לפי השאילתה הבא:

```
df.groupby('hour').size()\
  .plot.bar(title='Activity per hour')\
  .set_ylabel('Connection counts').get_figure().savefig('hours.png')
```

התוצאה:



אפשר לראות פה יחסית בבירור ששעות העבודה הן 08:00-17:00 יחד עם עובדים חרוצים שמתחילים לעבוד כבר ב-06:00 וכאלו שמסיימים לעבוד בסביבות חצות. השעות 06:00-00:00 הן שקטות כמעט לחלוטין ולכן אם נתמקד בהן נוכל אולי למצוא פעילות שמקורה בנוזקה. נכתוב שאילתא שתתן לנו את הכתובות הכי פעילות בתוך הארגון בשעות שהן לא שעות העבודה:

```
df[df['src_int'] & ~df['dst_int'] & ((df['hour'] >= 0) & (df['hour'] <
6))]\
  .groupby('dst_ip')\
  .size()\
  .pipe(lambda x: x[x > 0])\
  .sort_values(ascending=False).reset_index()
```

נריץ ונקבל:

```
0      148.26.83.117      3475
1      5.183.60.54        3467
2      251.139.203.226    355
3      250.42.245.153     330
4      8.8.8.8            279
5      234.8.152.205      218
6      55.107.247.62      51
7      24.161.197.98      34
8      198.70.42.36       12
9      15.254.213.206     5
10      80.213.210.227    1
11      65.64.19.34       1
```

```
12      21.91.160.44      1
13      112.146.59.67     1
14      242.138.45.182    1
```

ניתן לראות פה שתי כתובות שמקבלות מספר חריג של בקשות בשעות הללו. נבדוק אילו כתובות דיברו עם שני השרתים הללו:

```
df[df['src_int'] & ~df['dst_int'] & df['dst_ip'].isin(["148.26.83.117",
"5.183.60.54"]) & ((df['hour'] >= 0) & (df['hour'] < 6))]\
  .drop_duplicates("src_ip")["src_ip"]
```

נריץ ונקבל:

```
2       120.18.164.170
31      120.18.213.107
65      120.18.192.121
186     120.18.206.150
220     120.18.120.241
240      120.18.118.50
318      120.18.104.36
398     120.18.216.242
432      120.18.48.222
480      120.18.18.124
498      120.18.45.124
521       120.18.26.88
592       120.18.57.96
797     120.18.162.115
963      120.18.143.87
996     120.18.182.116
1005     120.18.110.15
1119     120.18.66.102
1242     120.18.23.222
1335      120.18.71.37
1353      120.18.88.93
1506    120.18.159.247
1614    120.18.178.138
1629     120.18.143.73
1775    120.18.218.147
1777    120.18.132.135
1856    120.18.173.154
1902    120.18.168.250
2005     120.18.157.13
2318     120.18.103.76
2385    120.18.205.198
2406      120.18.96.34
2416    120.18.171.175
2433     120.18.55.138
2460      120.18.9.187
2852     120.18.207.47
2988    120.18.195.179
3105     120.18.251.35
3187      120.18.27.78
3281     120.18.235.29
3314     120.18.202.14
3481    120.18.156.202
Name: src_ip, dtype: category
Categories (5497, object): ['24.133.117.23', '138.62.106.170', '92.150.22.192',
'209.69.254.129', ..., '173.119.192.16', '177.66.191.51', '178.195.62.138',
'120.121.209.92']
```

נגיש את הכתובות ונקבל את הדגל הראשון שלנו:

```
flag{hsjhjskdhjkhdjksd673276327sb}
```

דגל שני (150 נקודות)

כעת נבדוק מי שולח מחוץ לארגון כמויות גדולות של נתונים. השאילתה הראשונית שלנו תדרג את הכתובות הפנימיות ששולחות מידע החוצה לפי כמות המידע הנשלח:

```
df[df['src_int'] & ~df['dst_int']]\
  .groupby('src_ip')\
  .payload.sum()\
  .pipe(lambda x: x[x > 0])\
  .sort_values(ascending=False).head()
```

התוצאה:

```
src_ip
120.18.53.84      24795472
120.18.215.38     11774718
120.18.231.65      8447175
120.18.15.208      8047102
120.18.138.6       7872829
Name: payload, dtype: int64
```

ניתן לראות שהכתובת הראשונה שולחת מידע בפער ניכר מהכתובות האחרות. עם מי הנקודה הזו מתקשרת?

```
df[(df['src_ip'] == "120.18.53.84") &
~df['dst_int']][['dst_ip']].drop_duplicates("dst_ip").reset_index()
```

התוצאה:

```
      index     dst_ip
0      1896     8.8.8.8
```

זהו שרת DNS. האם מדובר בהברחת נתונים באמצעות DNS Tunneling? (עוד על כך ב[מאמר של יהודה גרסטל בגליון 20]).

נחפש את הכתובות שמדברות עם 120.18.53.84 בתוך הרשת הארגונית, התוצאה:

```
      src_ip      payload
17372911     120.18.212.32      450
5344676      120.18.252.47      450
8272425      120.18.179.89      449
337259       120.18.7.139       449
22821948     120.18.85.233      448
16895409     120.18.70.11       447
20668204     120.18.56.143      446
12639850     120.18.24.144      442
6659136      120.18.88.125      440
21480711     120.18.109.224     435
14669212     120.18.107.52      250
15580481     120.18.31.136      250
7722221      120.18.212.63      250
15651258     120.18.64.46       250
1333362      120.18.64.93       250
7723299      120.18.172.179     250
9784261      120.18.23.132      250
140100       120.18.39.9        250
15586973     120.18.92.185      250
1335597      120.18.115.245     250
```

אפשר לראות שבשלב מסויים מספר הפניות נופל מ-435 ל-250. נניח שזהו הגבול שבו אנו מפרידים בין פעילות זדונית לפעילות תמימה. אם כך, השרתים החשודים שלנו הם עשרת הראשונים. נגיש אותם ונקבל את הדגל השני שלנו:

```
flag{sjgdgyweo39834837421k2$%^344}
```

דגל שלישי (200 נקודות)

את הנוזקה השלישית נמצא בעזרת איתור קליקה גדולה ברשת שלנו, כלומר, אוסף כתובות גדול שבו נשלחה הודעה מכל כתובת לכל כתובת ללא יוצא מהכלל. מסתבר שזהו משהו שנוזקות עושות כאשר הן מייצרות רשת P2P פנימית, למשל כדי לצמצם את כמות התעבורה משרת מסוים או כדי להמנע מניהול ריכוזי של הבוטנט. קירוב לקליקה הגדולה ביותר ניתן למצוא באמצעות הקוד הבא:

```
import networkx
from networkx.algorithms.approximation.clique import large_clique_size
from collections import Counter

internal_edges_all = df[
  df['src_int'] & df['dst_int']
].drop_duplicates(['src_ip', 'dst_ip', 'dst_port'])
internal_ports = internal_edges_all.dst_port.unique()

port_upper_bounds = []
for p in internal_ports:
   internal_edges = internal_edges_all\
      .pipe(lambda x: x[x['dst_port'] == p])\
      .drop_duplicates(['src_ip', 'dst_ip'])
   edges = set()
   for l, r in zip(internal_edges.src_ip, internal_edges.dst_ip):
      k = min((l, r), (r, l))
      edges.add(k)
   degrees = Counter()
   for (l, r) in edges:
      degrees[l] += 1
      degrees[r] += 1
   max_clique_size = 0
   min_degrees = len(degrees)
   for idx, (node, degree) in enumerate(degrees.most_common()):
      min_degrees = min(min_degrees, degree)
      if min_degrees >= idx:
         max_clique_size = max(max_clique_size, idx+1)
      if min_degrees < max_clique_size:
         break
   port_upper_bounds.append((p, max_clique_size + 1))

max_port = 0
curr_max_clique = 0
max_clique_G = None
for p, max_clique_upper_bound in port_upper_bounds:
    if curr_max_clique > max_clique_upper_bound: break
    internal_edges = internal_edges_all\
        .pipe(lambda x: x[x['dst_port'] == p])\
        .drop_duplicates(['src_ip', 'dst_ip'])
    internal_nodes = set(internal_edges.src_ip) | set(internal_edges.dst_ip)
    G = networkx.Graph()
    G.add_nodes_from(internal_nodes)
```

```
    for l, r in zip(internal_edges.src_ip, internal_edges.dst_ip):
        G.add_edge(l, r)
    _size = large_clique_size(G)
    if curr_max_clique < _size:
        curr_max_clique = _size
        max_port = p
        max_clique_G = G

max_clique_ips =
networkx.algorithms.approximation.clique.max_clique(max_clique_G)
```

נריץ ונקבל:

```
{'120.18.123.198',
 '120.18.132.228',
 '120.18.14.179',
 '120.18.147.29',
 '120.18.15.122',
 '120.18.154.145',
 '120.18.158.121',
 '120.18.187.161',
 '120.18.201.129',
 '120.18.241.224',
 '120.18.254.217'}
```

נגיש ונקבל את הדגל השלישי:

```
flag{12124wdjhjh78882saslw@90817#4}
```

הדגל הרביעי (300 נקודות)

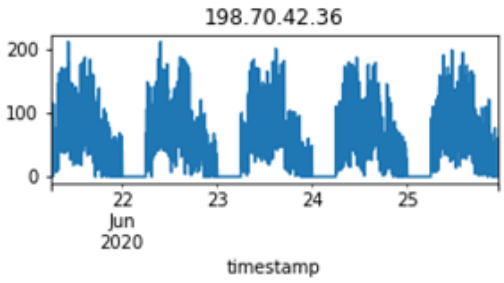
את הדגל הרביעי מצאנו כנראה במקרה. התחלנו מחיפוש אחר כתובת שמבצעת בקשות בזמנים קצובים, למשל כל חצי שעה או כל 5 דקות. הרצנו את השאילתה הבאה על מנת למצוא כתובות ששולחות הודעה בכל דקה:

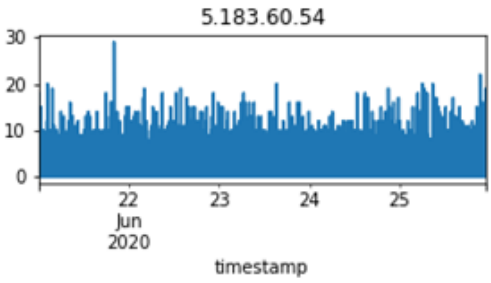
```
df[df['src_int'] & ~df['dst_int']]\
  .drop_duplicates(['dst_ip', 'minute'])\
  .groupby('dst_ip').size()\
  .pipe(lambda x: x[(x == 60)])
```

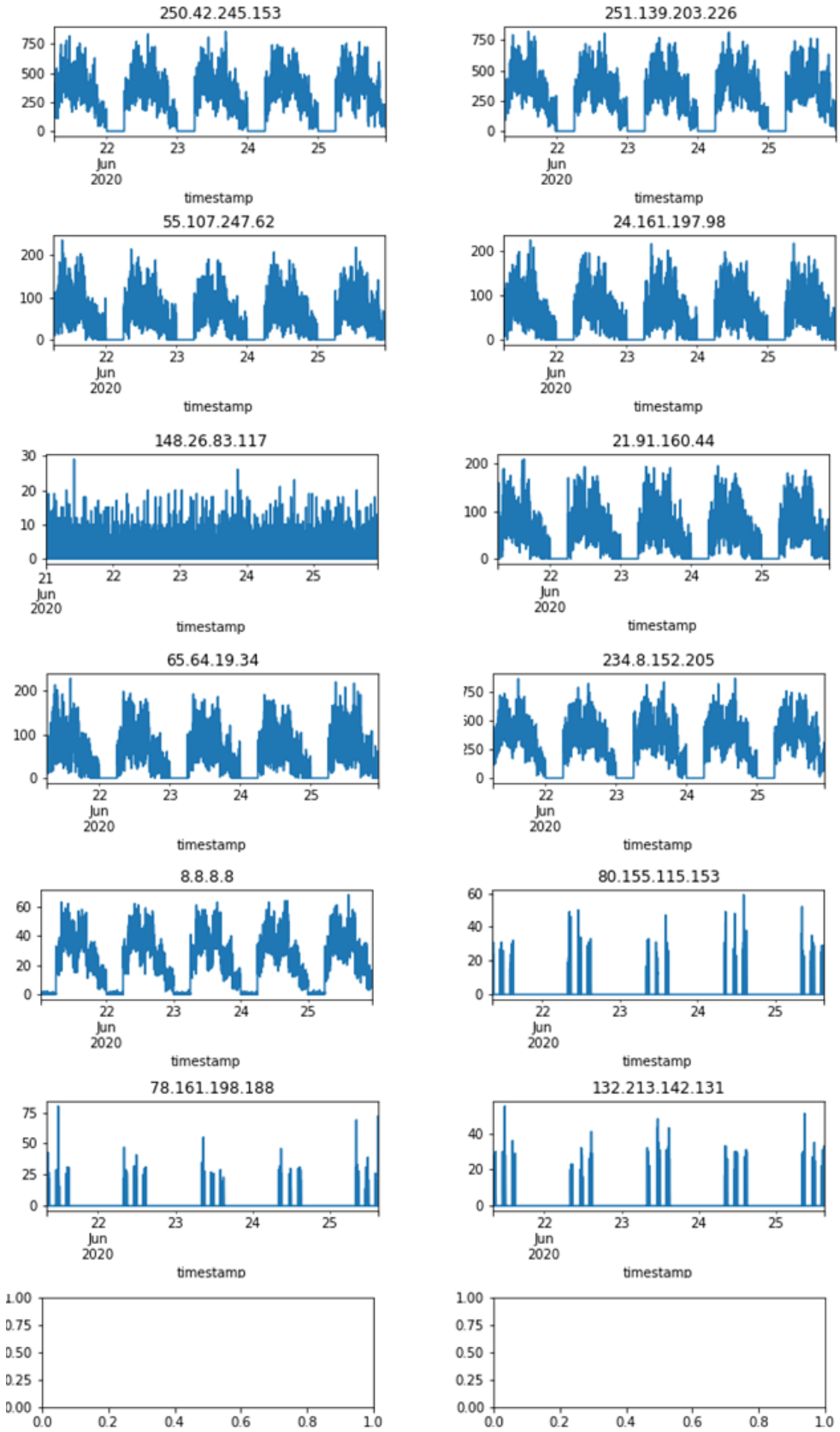
השאילתה החזירה 14 כתובות, ורצינו לראות את התוצאה בתור גרף:

```
import itertools

suspecious_ips = df[df['src_int'] & ~df['dst_int']]\
  .drop_duplicates(['dst_ip', 'minute'])\
  .groupby('dst_ip').size()\
  .pipe(lambda x: x[(x == 60)])\
  .index.tolist()

numl_cols = 4

num_rows = (len(suspecious_ips) // numl_cols) + 1

fig, axs = plt.subplots(ncols = numl_cols, nrows = num_rows,
figsize=(20, 10))
fig.tight_layout(pad=6.0)

for ax, ip in zip(itertools.chain(*axs), suspecious_ips):
    df[df['dst_ip']==ip]\
        .set_index('timestamp')\
        .resample('Min').size()\
        .plot(title = ip, ax = ax)
```

הגרפים שקיבלנו:






250.42.245.153
251.139.203.226
55.107.247.62
24.161.197.98
148.26.83.117
21.91.160.44
65.64.19.34
234.8.152.205
8.8.8.8
80.155.115.153
78.161.198.188
132.213.142.131
timestamp
22 Jun 2020
23
24
25
21 Jun 2020

אפשר לראות בבירור ששלושת הגרפים האחרונים (מבין אלו שיש להם נתונים) מציגים דפוסי התנהגות שונים לחלוטין מכל מה שבא לפני. שלושת הגרפים הללו מייצגים שטף של בקשות שנוצר בשלושה מקבצים מרוכזים ביום. נראה חשוד מאוד.

נמצא אילו כתובות פנימיות מקושרות לשלוש הכתובות שמצאנו:

```
df[(df.dst_ip.isin(['80.155.115.153', '78.161.198.188',
'132.213.142.131']))].drop_duplicates('src_ip')["src_ip"]
```

התוצאה:

```
503796    120.18.201.129
504115       120.18.2.84
515835    120.18.146.190
542064     120.18.216.16
567744       120.18.54.9
582629     120.18.102.12
623570    120.18.248.164
635570    120.18.123.205
647223    120.18.205.249
654354     120.18.138.14
658088    120.18.132.113
662042    120.18.188.217
666483     120.18.12.170
668357     120.18.202.49
689205     120.18.136.32
693631     120.18.219.23
717801    120.18.180.110
739725     120.18.240.47
747601    120.18.168.231
751304     120.18.36.143
757815    120.18.216.159
768456    120.18.184.225
771843    120.18.224.201
805606    120.18.102.141
817697    120.18.185.238
852189     120.18.162.99
Name: src_ip, dtype: category
Categories (5497, object): ['24.133.117.23', '138.62.106.170',
'92.150.22.192', '209.69.254.129', ..., '173.119.192.16',
'177.66.191.51', '178.195.62.138', '120.121.209.92']
```

נגיש ונקבל את הדגל הרביעי:

```
flag{!#fdfsd^&dfsdsds*sds(sajh336}
```

אתגר Statistics (קטגוריה: AI, 20 נקודות)

Imagine you are a data scientist working alongside Marie Curie; a Laurent and a physicist. Inspired by her work on radiation, she comes up with an idea for creating a new stochastic neural network layer entitled "RadiumDropout". The suggested layer behaves as follows:

For each K'th neuron, an eight-sided die is tossed K times. The dropout is activated with probability of 1 if and only if the sum of the K tosses is larger than 3^K, and a probability of 0 otherwise.

What is the probability that the first neuron is dropped?

פתרון:

זהו הראשון מבין צמד אתגרי סטטיסטיקה ששימשו לכל היותר בתור חימום קל. לפי הנוסחה שבתיאור האתגר, עבור הנוירון הראשון, מטילים פעם אחת קובייה עם 8 פאות. ה-dropout יתבצע אם ורק אם ערך ההטלה גבוה מ-3. לכן, אם ערך ההטלה הוא 3 / 2 / 1 אזי ה-dropout לא יתבצע, ואם הוא 7 / 6 / 5 / 4 / 8 אזי הוא יתבצע. כלומר, סיכוי של 5/8.

אתגר Statistics 2 (קטגוריה: AI, 20 נקודות)

Still refering to challenge Statistics1:

What is the maximum number of neurons that will be dropped?

פתרון:

שוב, שאלה קלילה לחימום. ראינו כבר שקיים תרחיש שבו הנוירון הראשון יופל. עבור הנוירון השני, סכום ההטלות צריך להיות גבוה מ-9 על מנת שהנוירון יופל, וזה כמובן אפשרי (למשל אם בשתי ההטלות נקבל 8). מה לגבי הנוירון השלישי? במקרה זה סכום ההטלות צריך להיות גבוה מ-27, אך גם אם בכל ההטלות נקבל את הערך הגבוה ביותר (שהוא 8) לא נצליח לקיים את התנאי. מכאן שהמספר המקסימלי הינו 2.

אתגר Vision (קטגוריה: AI, 250 נקודות)

During your research on image similarity methods, you discover a corpus made of images, only to find out that the original images were deleted by their creator and only their 512D feature vector was saved. You are now provided with a single such binary vector (a pickled numpy array), which was created from a ResNet18 model (PyTorch), pre-trained on the ImageNet corpus.

Your task is to identify the object that appeared in the original image.

לאתגר צורף קובץ image_embedding.npy.

פתרון:

לפי תיאור האתגר, קיבלנו לידינו וקטור המייצג תמונה של עצם כלשהו, ועלינו למצוא מהו עצם זה. כל מה שאנחנו יודעים על הוקטור הוא שמדובר במודל של ResNet18 (רשת עצבית מלאכותית בעלת 18 שכבות) שעברה אימון באמצעות מאגר התמונות של ImageNet (מאגר ענק של תמונות מתויגות לצרכי deep learning). הגישה שלנו לפתרון הייתה לעבור על הקטגוריות של ImageNet, לבחור מספר תמונות אקראיות מתוך כל קטגוריה, לחשב עליהן וקטור בצורה דומה ולבדוק כמה התוצאה דומה לוקטור שקיבלנו באתגר. לצורך כך, מצאנו קובץ JSON גדול עם קטגוריות מתוך ImageNet:

```
{
   "0": ["n01440764", "tench"],
   "1": ["n01443537", "goldfish"],
   "2": ["n01484850", "great_white_shark"],

}
```

עברנו על הקטגוריות אחת-אחת, ועבור כל אחת ביצענו שאילתה ב-ImageNet כדי לקבל חזרה את רשימת התמונות ששייכות לקטגוריה. למשל, עבור קטגוריית דגי הזהב נקבל רשימה של תמונות של דגי זהב, ומתוכן נבחר את עשרת התמונות הראשונות (שאינן שבורות).

עבור כל תמונה, ביצענו חישוב מקביל ליצירת וקטור המייצג את התמונה, והשוונו לוקטור המקורי. במידה והוקטורים היו מספיק דומים (מעל threshold מסוים שהגדרנו), הדפסנו את התוצאה. זהו הקוד עצמו (מימשנו אותו באמצעות שימוש ב-Multi-Processing כאשר החלקים השונים מתקשרים בינם לבין עצמם באמצעות Queues):

```
from pwn import *
from PIL import Image
from multiprocessing.dummy import Pool as ThreadPool

import os
import json
import torch
import torchvision
import torchvision.models as models
import numpy as np
```

```
import requests
import logging
import multiprocessing
import matplotlib.pyplot as plt

THRESHOLD = 0.7
OUTPUT_FOLDER = "output"
NUM_THREADS = 20
MAX_IMAGES_PER_LABEL = 10

# Load the pretrained model
model = models.resnet18(pretrained = True)

# Use the model object to select the desired layer
layer = model._modules.get('avgpool')

# Set model to evaluation mode
model.eval()

transforms = torchvision.transforms.Compose([
    torchvision.transforms.Resize(256),
    torchvision.transforms.CenterCrop(224),
    torchvision.transforms.ToTensor(),
    torchvision.transforms.Normalize(mean = [0.5, 0.5, 0.5], std = [0.5, 0.5,
0.5]),])

def get_vector(image):
    # Create a PyTorch tensor with the transformed image
    t_img = transforms(image)
    # Create a vector of zeros that will hold our feature vector
    # The 'avgpool' layer has an output size of 512
    my_embedding = torch.zeros(512)

    # Define a function that will copy the output of a layer
    def copy_data(m, i, o):
        my_embedding.copy_(o.flatten())                 # <-- flatten

    # Attach that function to our selected layer
    h = layer.register_forward_hook(copy_data)
    # Run the model on our transformed image
    with torch.no_grad():                               # <-- no_grad context
        model(t_img.unsqueeze(0))                       # <-- unsqueeze
    # Detach our copy function from the layer
    h.remove()
    # Return the feature vector
    return my_embedding

def download_images(input_queue, output_queue):
    while not input_queue.empty():
        key, id, label = input_queue.get()

        try:
            r = requests.get("http://www.image-
net.org/api/text/imagenet.synset.geturls?wnid={}".format(id))
        except Exception:
            continue

        added_images = 0
        for i, image_url in enumerate(r.text.split("\n")):
            if image_url == "":
                continue

            if added_images > MAX_IMAGES_PER_LABEL:
                break

            try:
```

```
                img = Image.open(requests.get(image_url, stream = True).raw)
                output_queue.put((key, id, label, i, img))
                added_images += 1
            except Exception:
                pass

def main():

    if not os.path.exists(OUTPUT_FOLDER):
        os.mkdir(OUTPUT_FOLDER)

    img_list_queue  = multiprocessing.Queue()
    result_queue    = multiprocessing.Queue()

    v = np.load("image_embedding.npy")
    cos = torch.nn.CosineSimilarity(dim = 0)

    # https://s3.amazonaws.com/deep-learning-models/image-
models/imagenet_class_index.json
    with open("imagenet_class_index.json", "r") as imagenet_class_index:
        labels = json.load(imagenet_class_index)
        for key, (id, label) in labels.items():
            img_list_queue.put((key, id, label))

    img_list_thread_pool = ThreadPool(NUM_THREADS, download_images,
(img_list_queue, result_queue))

    with log.progress('Searching for similar objects...') as p:
        while not img_list_queue.empty():
            try:
                key, id, label, i, img = result_queue.get(timeout = 60)
                p.status(f"{key}: Label '{label}', image #{i}")
                pic_vector = get_vector(img).numpy()

                cosine_similarity = cos(torch.from_numpy(v),
torch.from_numpy(pic_vector))

                if cosine_similarity > THRESHOLD:
                    log.info(f"Found similar object: {label}, similarity:
{cosine_similarity}")
                    np.save(f"{OUTPUT_FOLDER}/{label}_{i}_vector.npy",
pic_vector)
                    plt.scatter(range(v.shape[0]), v)
                    plt.scatter(range(pic_vector.shape[0]), pic_vector)
                    plt.savefig(f"{OUTPUT_FOLDER}/{label}_{i}_image.png")
                    plt.clf()
            except multiprocessing.queues.Empty:
                break
            except RuntimeError as e:
                pass

    img_list_thread_pool.close()
    img_list_thread_pool.join()

if __name__ == "__main__":
    main()
```

הפלט:

```
root@kali:/media/sf_CTFs/shabak/Vision# python3 solve.py
[?] Searching for similar objects...: 420: Label 'banjo', image #26
[*] Found similar object: assault_rifle, similarity: 0.7856280207633972
[*] Found similar object: assault_rifle, similarity: 0.7135688662528992
[*] Found similar object: assault_rifle, similarity: 0.7446579337120056
```

התוצאה שהתקבלה הייתה assault rifle.

אתגר NLP (קטגוריה: AI, 250 נקודות)

לאתגר צורף קובץ nlp_embedding.npy יחד עם התמונה הבאה:

פתרון:

כמו באתגר הקודם, גם במקרה הזה אנחנו מקבלים לידינו וקטור שהתקבל מביצוע חישוב כלשהו על אובייקט, כאשר בפעם הקודמת מדובר היה בתמונה והפעם במשפט מתוך טקסט. אנחנו מקבלים את הפרטים על המודל בו השתמשו לביצוע החישוב (bert-base-multilingual-uncased), יחד עם הסבר כיצד להגיע מהמידול אל הוקטור עצמו (חישוב ממוצע על פני מספר שכבות נסתרות). מהמידע הזה, עלינו למצוא את המשפט המקורי, וממנו להגיע לאינדקס שלו בטקסט ("מספר השורה"). זה נשמע די מעורפל אבל הכל יתבהר ברגע שנצליח לפצח את הרמז.

בתור רמז, אנחנו מקבלים תמונה של הלוגו של השב"כ. אחד הדברים שכדאי לבדוק במהלך CTF כאשר מקבלים תמונה שלא ברור מה עוד אפשר לעשות איתה הוא לחפש בה מסרים שהוסתרו באמצעות סטגנוגרפיה. ישנם כלים ואתרים רבים שמסייעים לבצע חיפוש בצורה אוטומטית, למשל זה. אם נעלה אליו את התמונה, נגלה שהוסתר בה מסר חבוי באמצעות שימוש בביט התחתון של כל פיקסל:

נחפש את הטקסט הנסתר البَقَرَة בגוגל ונגלה שמדובר ב"סורת אל-בקרה" - הפרק השני בספר הקוראן שהוא במקרה גם הארוך ביותר בו (286 פסוקים). התובנה הזו מסבירה מספר דברים בתיאור התרגיל: מדוע השתמשו במודל רב-שפתי (multilingual), מה הייתה הכוונה ב"תווים המיוחדים" (ניקוד) וכיצד יצא שהטקסט ממוספר (מספרי פסוקים). אם כך, ברור לחלוטין מה עלינו לעשות כעת. עלינו לחשב וקטור עבור כל אחד מהפסוקים, ולאתר את הפסוק שעבורו הוקטור הינו הקרוב ביותר לוקטור שקיבלנו. הדבר נעשה באמצעות הקוד הבא:

```
from pwn import *
import numpy as np
from scipy import spatial
from transformers import BertTokenizer, BertModel, BertConfig
import os

tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-uncased')
config    = BertConfig.from_pretrained("bert-base-multilingual-uncased",
output_hidden_states = True)
model     = BertModel.from_pretrained("bert-base-multilingual-uncased", config =
config)
VECTOR_LEN = 768

def calc_layers_average(layers):
    res = np.zeros(VECTOR_LEN)
    count = 0
    for layer in layers:
        for level1 in layer:
            for level2 in level1:
                assert(len(level2) == VECTOR_LEN)
                res += level2.detach().numpy()
                count += 1
    res /= count
    return res

def calc_score(text):
    inputs = tokenizer(text, return_tensors="pt")
    outputs = model(**inputs)
    hidden_states = outputs[2]
    embedding_output = hidden_states[0]
    attention_hidden_states = hidden_states[1:]
    return calc_layers_average(attention_hidden_states)

scores = []
with log.progress('Calculating scores...') as p:
    with open("AlBaqarah.txt", encoding = "utf8") as f:
        line_num = 0
        for line in f:
            line = line.strip()
            if line != "":
                p.status(f"Working on line #{line_num}")
                scores.append(calc_score(line))
                line_num += 1

with log.progress('Searching for best match...'):
    data = np.load("nlp_embedding.npy")
    tree = spatial.KDTree(scores)
    res = tree.query(data)

log.success(f"Best match: line #{res[1] + 1}")
```

נריץ ונקבל שפסוק 147 הינו הקרוב ביותר.

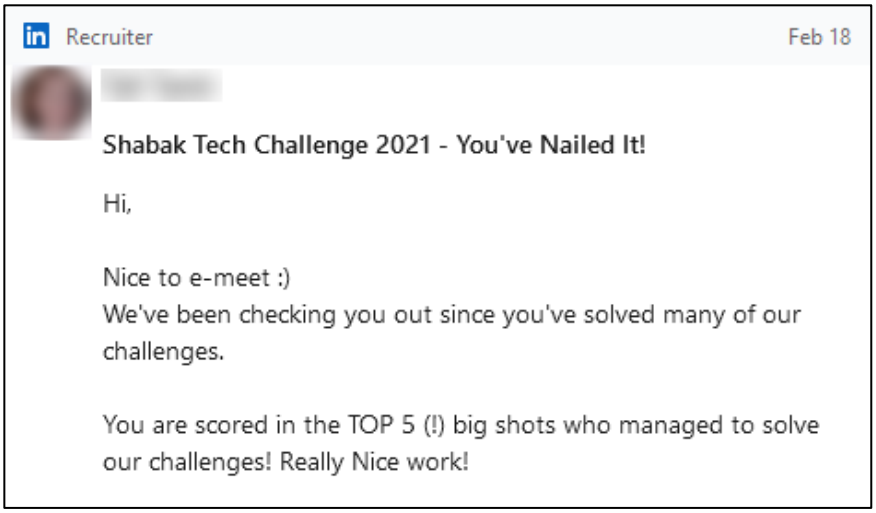
אתגר Speech (קטגוריה: AI, 500 נקודות)

האתגר האחרון בקטגוריית AI עסק בשחזור של קובץ שמע מתוך מערך של numpy. את האתגר לא הצלחנו לפתור במהלך ה-CTF ולכן אנו מזמינים אתכם לקרוא את הפתרון המצוין של Ariel Demidov בקישור הזה. כל הכבוד!

סיכום

ללא ספק, השב"כ הקפיצו השנה את הרמה וסיפקו CTF מגוון, מהנה ומאתגר מאוד. באופן אירוני למדי, המאמר הזה כולל דווקא פתרונות לתחומים שבהם אפשר לומר שעשינו את צעדינו הראשונים תוך כדי ה-CTF, מכיוון שעבור התחומים הקלאסיים יותר התפרסמו כבר פתרונות בגליונות קודמים. שמחנו מאוד שהצלחנו לפתור אתגרים מתחומים שחדשים לנו, אבל עם זאת קיים סיכוי סביר שחלק מהפתרונות שלנו לא היו אופטימליים ומומחה או מומחית בתחום היו בוחרים לפתור אותם בדרך אחרת.

מבחינת אתגרים, האתגר המוצלח והמיוחד ביותר היה כנראה NFC (קטגוריית רברסינג) אבל החלק המגניב והמפתיע ביותר הגיע בדיעבד כשקיבלנו כל אחד לחשבון הלינקדאין האישי את ההודעה הבאה:


Recruiter Feb 18

Shabak Tech Challenge 2021 - You've Nailed It!

Hi,

Nice to e-meet :)
We've been checking you out since you've solved many of our challenges.

You are scored in the TOP 5 (!) big shots who managed to solve our challenges! Really Nice work!

מה שמפתיע פה הוא שבמהלך ההרשמה ל-CTF לא סיפקנו מייל או פרטים שמקושרים באופן ישיר לפרופיל שלנו, אלא כינויים וכתובות מייל משניים, ונדרשה פה "עבודת שב"כ" מסויימת על מנת להגיע משם אל הפרופילים הפומביים. Well played!

כל הכבוד ליוצרי האתגר, מצפים מאוד לאתגר של שנה הבאה!