WOJSKOWA AKADEMIA TECHNICZNA
im. Jarosława Dąbrowskiego
WYDZIAŁ CYBERNETYKI

Big Data – projekt z zakresu bezpieczeństwa informacji

Temat: „Analiza cyberzagrożeń - system wczesnego informowania (CTI)”

1. Opis tematu

1.1. Temat i Cel

Temat naszego projektu to „Analiza cyberzagrożeń - system wczesnego informowania (CTI)". Za cel projektu postawiliśmy sobie przygotowanie środowiska, którego zadaniem będzie pobieranie danych, przetwarzanie ich, zapisywanie, analizowanie oraz prezentowanie wyników analiz w postaci graficznej – za pomocą wykresów, tabel, itp. Obszar, w którym postanowiliśmy się poruszać to cyberzagrożenia, a więc interesowały nas wszelakie informacje, które mogłyby informować o zagrożeniach cybernetycznych ze świata – między innymi niebezpiecznych (złośliwych) adresach ip, niebezpiecznych url'ach phishingowych.

2. Wybrane technologie:

W projekcie zostały wykorzystane technologie takie jak:

- Baza danych elasticsearch (na pięciu węzłach)
- Baza danych neo4j (na jednym węźle)
- Apache Spark
- Skrypty pisane w języku Python
- Kibana

3. Wykorzystywane zbiory danych

Jako ciekawe źródło wykorzystywane na wstępnych etapach analizy posłużył nam poniższy link:

https://github.com/hslatman/awesome-threat-intelligence

Opisano w nim różnego rodzaju zbiory danych (między innymi api dostępne z internetu) prezentujące informacje na temat cyberzagrożeń.

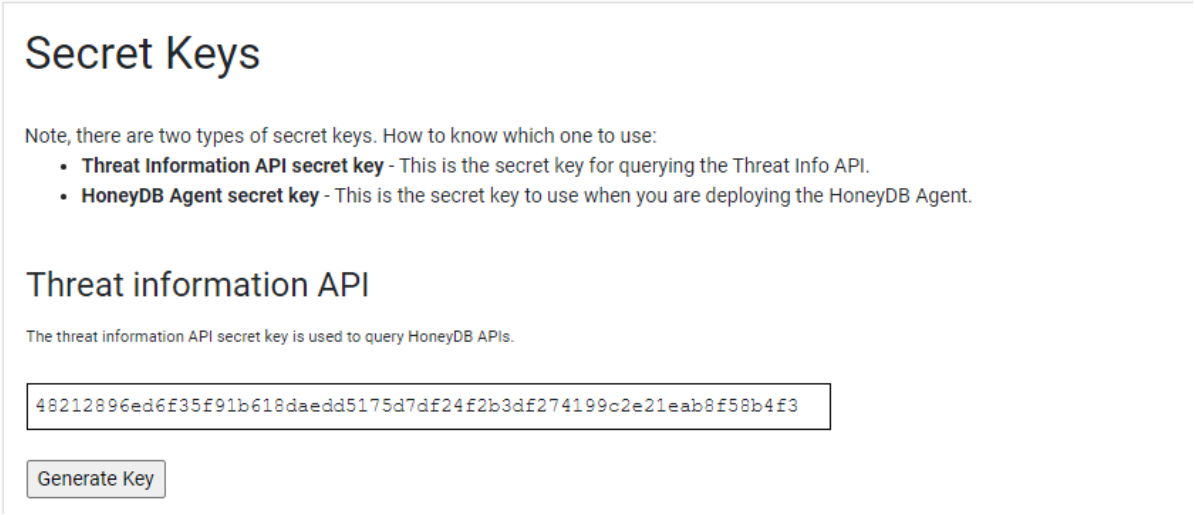
3.1. Zbiór danych HoneyDB

HoneyDB – jest to baza informacji zbieranych przez specjalnie wystawione do tego celu serwery (tzw. honeypots), które mają za zadanie przyciągnąć do siebie atakujących, a następnie zebrać informację na ich temat. W ten sposób baza HoneyDB zbiera dane na temat:

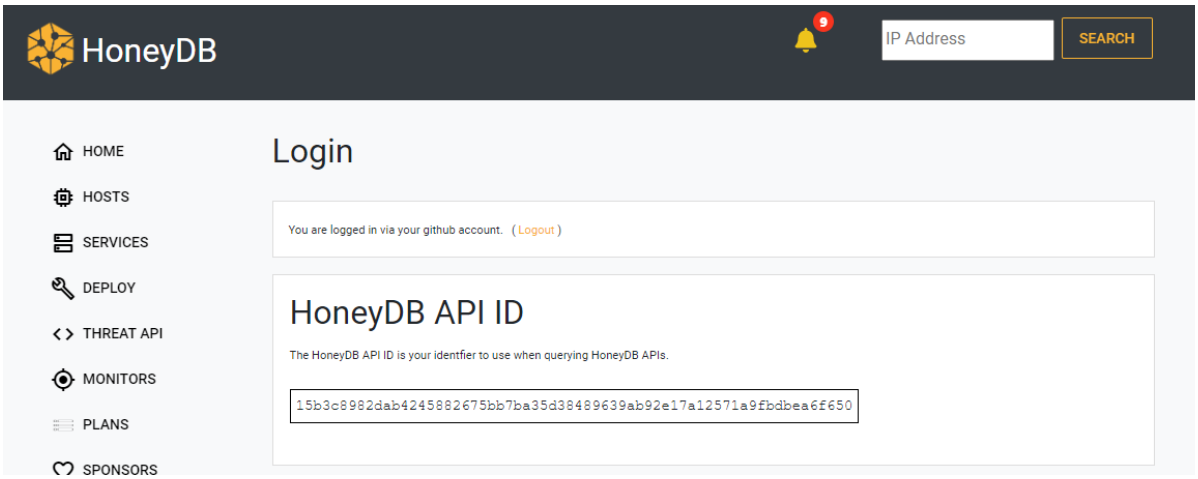
- Złośliwych adresów ip
- Liczby prób połączeń w ciągu dnia
- Częstości ataków na dane usługi

Zbiór danych wystawia specjalnie przygotowane API, aby umożliwić użytkownikom pobieranie danych o atakujących. API jest zabezpieczone, więc aby uzyskać do niego

dostęp należy wygenerować specjalny klucz, a następnie wykorzystywać klucz do uwierzytelnienie się podczas wysyłania requestów do API.



Rysunek 1 Generowanie klucza do API



Rysunek 2 Odczytanie id wykorzystywanego do połączenia z API

Tak jak wspomniano powyżej, zbiór danych HoneyDB zawiera informacje o złośliwych adresach IP, które próbowały się podłączyć do wystawionych honeypotów. Informacje te

prezentuje część API o nazwie: Bad Hosts. Poniżej zaprezentowano oficjalne informacje ze strony https://honeydb.io/api/bad-hosts na temat tej części.

Bad Hosts

A bad host is a host on the Internet that has connected or attempted to connect to one of the honeypots that feed data to HoneyDB. In general, there is no legitimate reason for any host to connect to these honeypots. So those that do can be considered bad, and a potential threat. If you see connectivity from any of these hosts on your network it may be malicious and may require some investigation.

Only the last 24 hours of bad host data is made available.

URL:

https://honeydb.io/api/bad-hosts

Example Request:

```
curl --header "X-HoneyDb-ApiId: <enter your api_id here>" \
        --header "X-HoneyDb-ApiKey: <enter your api_key here>" \
        https://honeydb.io/api/bad-hosts
```

The response is provided in JSON format and consists of the following fields:

- **remote_host** - The IP address of the bad host.
- **count** - The number of connections made by the bad host.
- **last_seen** - The date of the connection made by the bad host.

Example Response:

```
[{"remote_host":"121.183.78.86","count":"203","last_seen":"2015-09-07"},
{"remote_host":"117.12.127.121","count":"203","last_seen":"2015-09-07"},
...
{"remote_host":"60.3.51.115","count":"203","last_seen":"2015-09-07"}]
```

Rysunek 3 Informacje na temat Bad Hosts

Z kolei informacje o łącznej liczbie prób połączenia do każdego z wystawionych serwisów w ciągu jednej doby prezentuje część API o nazwie: Services. Poniżej zaprezentowano oficjalne informacje ze strony https://honeydb.io/api/services na temat tej części.

Services

Services are the network protocols emulated by honeypot sensors.

Only the last 24 hours of services data is made available.

URL:

https://honeydb.io/api/services

Example Request:

```
curl --header "X-HoneyDb-ApiId: <enter your api_id here>" \
    --header "X-HoneyDb-ApiKey: <enter your api_key here>" \
    https://honeydb.io/api/services
```

The response is provided in JSON format and consists of the following field:

- **service** - The name of the service.
- **count** - The number of events associated with the service name.

Example Response:


```
[
    {
        "service": "VNC",
        "count": "1702004"
    },
    {
        "service": "SSH",
        "count": "177504"
    },
```


Rysunek 4 Informacje na temat Services

3.2.Zbiór danych PhishTank

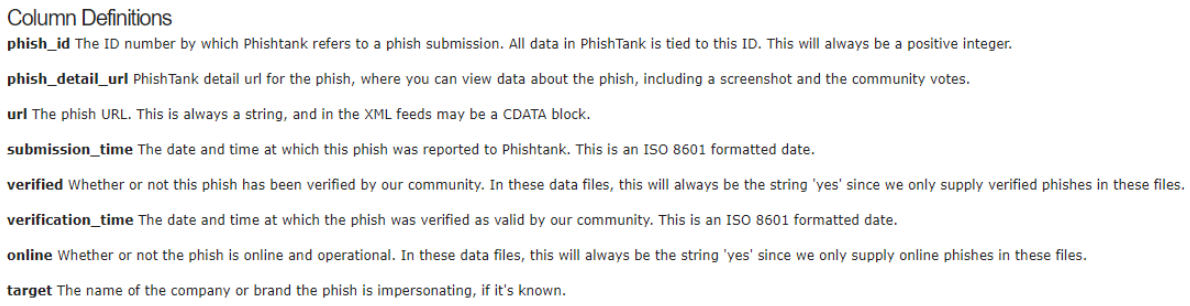
PhishTank – jest to baza informacji na temat niebezpiecznych adresów URL, wykorzystywanych jako adresy phishingowe. Dane zbierane są przez specjalistów z całego

świata. Do PhishTank trafiają zgłoszone adresy URL, a następnie społeczność dokonuje ich weryfikacji.

W ten sposób powstaje bardzo duża, zweryfikowana baza zawierające dane na temat:

- Jakie adresy URL są niebezpieczne
- Kiedy ostatni ras zauważono niebezpieczny URL
- Kiedy dokonano jego weryfikacji
- Z jakiego kraju pochodził URL (konkretnie w jakim kraju zarejestrowana była domena)
- Czy strona internetowa jest online (aktualnie)
- Pod jaką stronę się podszywa podejrzany URL

Podsumowując, zbiór danych PhishTank zawiera informacje o złośliwych adresach URL, które były wykorzystywane w kampaniach phishingowych. Poniżej zaprezentowano oficjalne informacje ze strony https://www.phishtank.com/


Column Definitions

phish_id The ID number by which Phishtank refers to a phish submission. All data in PhishTank is tied to this ID. This will always be a positive integer.

phish_detail_url PhishTank detail url for the phish, where you can view data about the phish, including a screenshot and the community votes.

url The phish URL. This is always a string, and in the XML feeds may be a CDATA block.

submission_time The date and time at which this phish was reported to Phishtank. This is an ISO 8601 formatted date.

verified Whether or not this phish has been verified by our community. In these data files, this will always be the string 'yes' since we only supply verified phishes in these files.

verification_time The date and time at which the phish was verified as valid by our community. This is an ISO 8601 formatted date.

online Whether or not the phish is online and operational. In these data files, this will always be the string 'yes' since we only supply online phishes in these files.

target The name of the company or brand the phish is impersonating, if it's known.

Rysunek 5 Informacje na temat PhishTank

Zbiór danych wystawia specjalnie przygotowane API, aby umożliwić użytkownikom pobieranie danych o złośliwych adresach URL. Podobnie jak w przypadku Bad Hosts, API jest zabezpieczone, więc aby uzyskać do niego dostęp należy wygenerować specjalny klucz, a następnie wykorzystywać klucz do uwierzytelnienie się podczas wysyłania requestów do API. O klucz można zawnioskować na stronie: https://www.phishtank.com/developer_info.php

```
"phish_id" : "7977469",
"url" : "https://wvvwdaviviendaportalpersonas.daviviendaportl.repl.co/",
"phish_detail_url" : "http://www.phishtank.com/phish_detail.php?phish_id=7977469",
"submission_time" : "2022-12-14T20:15:35+00:00",
"verified" : "yes",
"verification_time" : "2022-12-14T20:42:16+00:00",
"online" : "yes",
"details" : [
  {
    "ip_address" : "34.149.204.188",
    "cidr_block" : "34.144.0.0/13",
    "announcing_network" : "15169",
    "rir" : "arin",
    "country" : "US",
    "detail_time" : "2022-12-14T20:22:16+00:00"
  }
],
"target" : "Other"
```


Rysunek 6 Tak prezentują się pobrane dane z PhishTank dotyczące jednego rekordu i wyświetlone w Kibanie

4. Model architektury

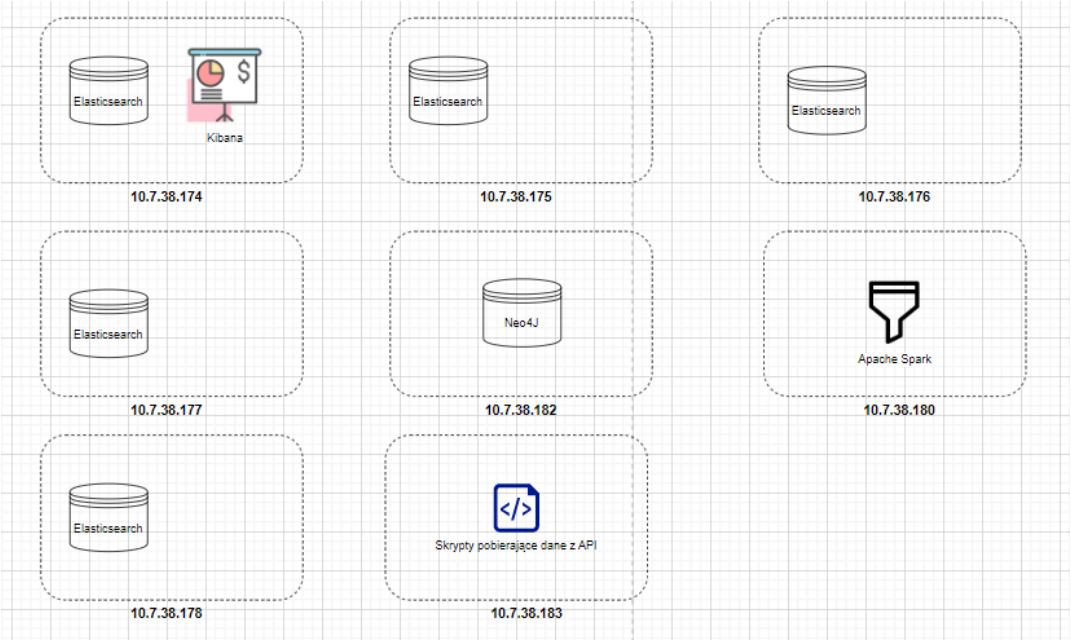
Niniejszy rozdział poświęcony jest architekturze. Pokazane zostały przepływy danych między różnymi wykorzystanymi komponentami. Do tego wskazano, który komponent znalazł się na danej z przydzielonych maszyn wirtualnych.

4.1.Przydzielone maszyny wirtualne

Poniższa tabela prezentuje przydzielone naszej grupie maszyny wirtualne, wypisana została ich adresacja oraz co znalazło się na każdej z maszyn wirtualnych.

Nazwa	Stan	Adres IP	Co znajdzie się na maszynie
rpd-11	Powered On	10.7.38.174	Elasticsearch, Kibana
rpd-12	Powered On	10.7.38.175	Elasticsearch
rpd-13	Powered On	10.7.38.176	Elasticsearch
rpd-14	Powered On	10.7.38.177	Elasticsearch
rpd-15	Powered On	10.7.38.178	Elasticsearch
rpd-16	Powered On	10.7.38.179	
rpd-17	Powered On	10.7.38.180	Apache Spark, kod przetwarzający dane z Elastic do Spark
rpd-18	Powered On	10.7.38.181	
rpd-19	Powered On	10.7.38.182	Neo4J
rpd-20	Powered On	10.7.38.183	Skrypty pobierające i oczyszczające dane z API

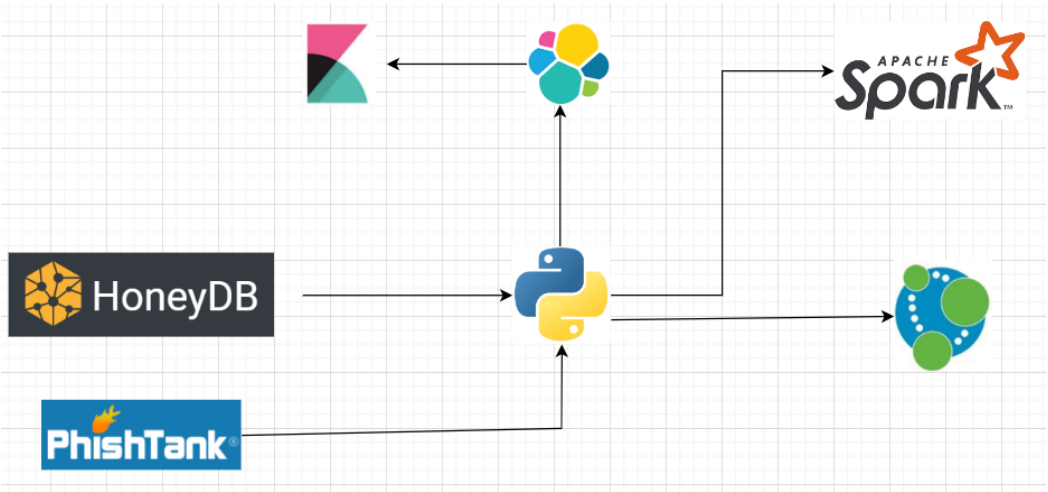
Dodatkowo prezentujemy zawartość powyższej tabeli na poniższym diagramie.



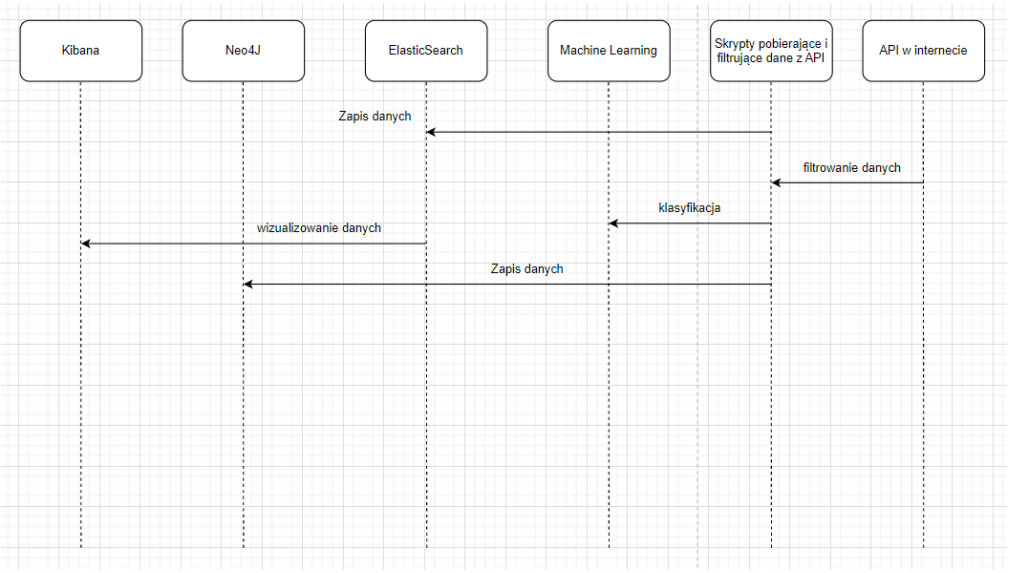
Rysunek 7 Architektura rozwiązania

4.2.Przepływy danych

Poniższe dwie ilustracje prezentują przepływy danych w naszym projekcie zespołowym. Zgodnie z pierwszym diagramem na początek pobierane są z 2 różnych API: HoneyDB, PhishTank. Następnie dane przetwarzane są poprzez skrypty w pythonie i trafiają do Sparka, ElasticSearch oraz Neo4J. Za pomocą Kibany prezentujemy przygotowane dashboardy.



Rysunek 8 Przypływy danych - loga



Rysunek 9 Przepływy danych - diagram

5. Przetwarzanie danych

Przetwarzanie następuje poprzez skrypty umieszczone na maszynie o adresie ip: 10.7.38.183. Na potrzeby realizacji zadania napisano dwa różne skrypty, pierwszy pobiera, przetwarza i zapisuje dane z HoneyDB, drugi pobiera, przetwarza i zapisuje dane z PhishTank. W skryptach wykorzystano bibliotekę requests, aby pobierać dane z API oraz aby zapisywać dane do baz danych za pomocą metody http POST. Dodatkowo dla bazy danych HoneyDB wykorzystywana jest biblioteka datetime celem dodania do rekordów daty dnia, z którego dane rekordy pochodzą.

Poniżej zaprezentowano skrypt dla HoneyDB:

```python
import requests
import json
import datetime
from datetime import date
import schedule

counter = 0


def getData(url, headers):
    #print("pobieram dane")
    r = requests.get(url, headers = headers)
    dataJson = r.text
    data = json.loads(dataJson)
    #print(data)
    return data


def saveToElastic(data, counter, index):
    #print("rozpoczynam zapis do elastica")
    for x in data:
        r = requests.post(url="http://10.7.38.174:9200/" + index + "/_doc/" + str(counter),
                          headers={"kbn-xsrf": "reporting", 'Content-Type': 'application/json'}, data=json.dumps(x),
                          verify=False)
        if r.status_code == 201 or r.status_code == 200:
            #print("Data saved!")
            counter = counter + 1
        else:
            print("cant save data,  body: " + r.text)

    print(" Data inserting complited")
    return counter


url_bad_hosts = 'https://honeydb.io/api/bad-hosts'
filename_bad_hosts = 'bad_hosts.txt'
index_bad_hosts = 'honeydb_bad_hosts'

url_services = 'https://honeydb.io/api/services'
```

```python
filename_services = 'services.txt'
index_services = 'honeydb_services'

headers = {'X-HoneyDb-ApiId': '15b3c8982dab4245882675bb7ba35d38489639ab92e17a12571a9fbdbea6f650',
           'X-HoneyDb-ApiKey': '4821289oed6f35f91b618daedd5175d7df24f2b3df274199c2e21eab8f58b4f3'}

#es = Elasticsearch('http://10.7.38.174:9200')
#es.info()
#es.indices.create(index= 'honeydb-bad-hosts')

def job():
    #print("rozpoczynam dzialanie")
    f = open(filename_bad_hosts, "r")
    counter = int(f.read())
    response = getData(url_bad_hosts, headers)
    counter = saveToElastic(response, counter, index_bad_hosts)
    f = open(filename_bad_hosts, "w")
    f.write(str(counter))
    f.close()
    return

def add_date_to_response(response):
    today = date.today()
    d1 = today.strftime("%Y-%m-%d")
    for i in response:
        i["date"] = d1
    return response

def job2():
    #print("rozpoczynam dzialanie")
    f = open(filename_services, "r")
    counter = int(f.read())
    response = getData(url_services, headers)
    response = add_date_to_response(response)
    counter = saveToElastic(response, counter, index_services)
    f = open(filename_services, "w")
```

```
    f.write(str(counter))
    f.close()
    return

schedule.every().day.at("12:00").do(job)
schedule.every().day.at("23:59").do(job2)
#schedule.every(30).seconds.do(job)

while True:
    schedule.run_pending()
    #time.sleep(60) # wait one minute

```

Poniżej zaprezentowano skrypt dla PhishTank:

```
import requests
import json
import schedule

def get_number_of_hits_elastic():
    r = requests.get(url="http://10.7.38.174:9200/phishtank/_count",
                     headers={"kbn-xsrf": "reporting", 'Content-Type': 'application/json'}
                     , verify=False)
    if r.status_code == 201 or r.status_code == 200:
        dataJson = r.text
        data = json.loads(dataJson)
        count = data.get("count")
        print("Number of records:")
        print(count)
    else:
        print("cant get value: " + r.text)
    return count


def get_table_of_phish_id(number_of_records):
    list_of_phish_id_already_in_database = []
    for x in range(1,number_of_records+1):
        r = requests.get(url="http://10.7.38.174:9200/phishtank/_doc/" + str(x),
                         headers={"kbn-xsrf": "reporting", 'Content-Type': 'application/json'}#, data=json.dumps(x),
                         ,verify=False)
        if r.status_code == 201 or r.status_code == 200:
            dataJson = r.text
            data = json.loads(dataJson)
            data_from_phishTank = data.get("_source")
            phish_id = data_from_phishTank.get('phish_id')
            list_of_phish_id_already_in_database.append(phish_id)
        else:
            print("cant get phish_id: " + r.text)

    return list_of_phish_id_already_in_database


headers = {'User-Agent': 'phishtank'}

```

```python
url = "http://data.phishtank.com/data/online-valid.json"

counter = 1

def getData(url):
    r = requests.get(url)
    dataJson = r.text
    data = json.loads(dataJson)
    return data

def saveToElastic(data, counter, table_of_phish_id):
    for x in data:
        if x.get("phish_id") not in table_of_phish_id:
            r = requests.post(url="http://10.7.38.174:9200/phishtank/_doc/" + str(counter),
                              headers={"kbn-xsrf": "reporting", 'Content-Type': 'application/json'}, data=json.dumps(x),
                              verify=False)
            if r.status_code == 201 or r.status_code == 200:
                counter = counter + 1
            else:
                print("cant save data,  body: " + r.text)

    print(" Data inserting complited")



def job():
    number_of_records = get_number_of_hits_elastic()
    table_of_phish_id = get_table_of_phish_id(number_of_records)
    saveToElastic(getData(url), number_of_records+1, table_of_phish_id)
    return

schedule.every().day.at("12:07").do(job)


while True:
    schedule.run_pending()
```

Na samym końcu skryptów wykorzystywana jest biblioteka schedule, aby zaplanować cykliczne wykonywanie się kodu, o danej godzinie.

6. Realizacja

6.1. Instalacja Apache Spark

Instalacja i konfiguracja Apache Spark

Ściągnięcie i zainstalowanie Apach spark oraz skonfigurowanie środowiska

```
[root@rpd-17 ~]# wget https://dlcdn.apache.org/spark/spark-3.3.1/spark-3.3.1-bin-hadoop3.tgz
--2023-01-22 13:53:17--  https://dlcdn.apache.org/spark/spark-3.3.1/spark-3.3.1-bin-hadoop3.tgz
Resolving dlcdn.apache.org (dlcdn.apache.org)... 151.101.2.132, 2a04:4e42::644
Connecting to dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 299350810 (285M) [application/x-gzip]
Saving to: ‘spark-3.3.1-bin-hadoop3.tgz’

100%[=========================================================================================================>]

2023-01-22 13:53:20 (91.7 MB/s) - ‘spark-3.3.1-bin-hadoop3.tgz’ saved [299350810/299350810]

[root@rpd-17 ~]# ls
anaconda-ks.cfg  spark-3.3.1-bin-hadoop3.tgz
[root@rpd-17 ~]# tar -xzf spark-3.3.1-bin-hadoop3.tgz
[root@rpd-17 ~]# ls
anaconda-ks.cfg  spark-3.3.1-bin-hadoop3  spark-3.3.1-bin-hadoop3.tgz
[root@rpd-17 ~]# export SPARK_HOME=$HOME/spark-3.3.1-bin-hadoop3
[root@rpd-17 ~]# export PATH=$PATH:$SPARK_HOME/bin
[root@rpd-17 ~]# echo 'export SPARK_HOME=$HOME/^C
[root@rpd-17 ~]# echo 'export SPARK_HOME=$HOME/spark-3.3.1-bin-hadoop3' >> .bash_profile
[root@rpd-17 ~]# ^C
[root@rpd-17 ~]# echo 'export PATH=$PATH:$SPARK_HOME/bin' >> .bash_profile
```

Sprawdzenie poprawności instalacji apacheSpark poprzez uruchomienie pyspark

```
[root@rpd-17 bin]# pyspark
Python 3.6.8 (default, Nov 16 2020, 16:55:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
23/01/22 15:55:22 WARN Utils: Your hostname, rpd-17 resolves to a loopback address: 127.0.1.1; using 10.7.38.180 instead (on interface ens192)
23/01/22 15:55:22 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/01/22 15:55:23 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.3.1
      /_/

Using Python version 3.6.8 (default, Nov 16 2020 16:55:22)
Spark context Web UI available at http://10.7.38.180:4040
Spark context available as 'sc' (master = local[*], app id = local-1674399324447).
SparkSession available as 'spark'.
>>>
Traceback (most recent call last):
  File "/root/spark-3.3.1-bin-hadoop3/python/pyspark/context.py", line 363, in signal_handler
    raise KeyboardInterrupt()
KeyboardInterrupt
>>>
```

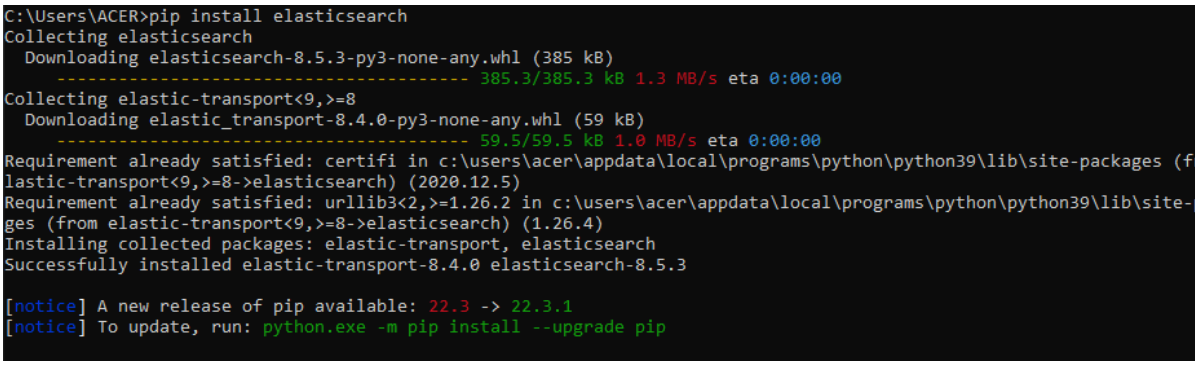
Instalacja python3

Instalacja java 1.8 openJDK

Przeniesienie plików z pomocą WinScp

6.2. Instalacja ElasticSearch

Klaster Elasticsearch oraz Kibana zostały uruchomione na 5 maszynach wirtualnych przy wykorzystaniu narzędzia Docker. Kibana znajduje się na jednym z węzłów: http://10.7.38.174:5601/app/kibana#/home .



6.3. Uczenie maszynowe

Przetwarzanie danych za pomocą uczenia maszynowego

Nasz zaimplementowany model uczenia maszynowego ma za zadanie, na postawie podanego adresu url, stwierdzić czy powinien być on sklasyfikowany jako Phishing (niebezpieczny adres, który prawdopodobnie ma na celu wyłudzenie poufnych danych lub zainfekowania szkodliwym oprogramowaniem), czy też można go uznać za bezpieczny.

Aby zdobyć niebezpieczne adresy url, model wykorzystuje dane z bazy Elastic postawionej na maszynie 10.7.38.174 – dane pobrane z opisanej wcześniej strony PhishTank, która zawiera informacje o zidentyfikowanych phishingach. Przy pomocy skryptu zebrano 1000 rekordów z bazy, równocześnie dzieląc i wyodrębniając potrzebne wartości do elementów modelu.

„Bezpieczne" adresy url zostały wygenerowane sztucznie za pomocą skryptu, który generował losowe znaki do narzuconych wcześniej domen np. facebook, youtube, onet, sharepoint

Model przyjmuje dane:

„https" – może przyjąć wartości „0" gdy podany url nie korzysta z protokołu/ma go w nazwie lub „1" w przeciwnym przypadku,

„subdomain_chars" - liczba znaków subdomeny podanej w adresie url,

„domain_chars" - liczba znaków domeny podanej w adresie url,

„commas" - liczba kropek w podanym adresie url,

„php" - przyjmuje wartości „1" gdy adres zawiera „php" i „0" gdy nie zawiera,

„url_length" - długość adresu url ,

„phish" - tutaj wartość „1" jest dla adresów url pobranych z PhishTank, a „0" dla wygenerowanych 'bezpiecznych' adresów url

```
import pandas as pd
dane = pd.read_json('mix3.json')

dane
```

	https	subdomain_chars	domain_chars	commas	php	url_length	country	phish
0	1	10	10	2	0	51	PL	0
1	1	10	10	2	0	51	PL	0
2	1	10	10	2	0	51	PL	0
3	1	10	10	2	0	51	PL	0
4	1	10	10	2	0	51	PL	0
...	...	...	...	...	...	...	...	...
1990	1	12	5	3	0	85	BR	1
1991	1	13	5	2	0	34	US	1
1992	0	7	7	2	0	29	HK	1
1993	1	0	4	1	0	21	US	1
1994	0	7	7	2	0	29	HK	1

1995 rows × 8 columns

Budując model skorzystaliśmy z bibliotek PySparka w zakresie Uczenia Maszynowego, a dokładniej meotdy klasyfikacji binarnej - Regresji Logistycznej.

```
[163] assembler = VectorAssembler(inputCols=['https','subdomain_chars', 'domain_chars', 'commas', 'php', 'url_length'],outputCo
      dataFrame = assembler.transform(dataFrame)
```

```
[136] from pyspark.ml.classification import LogisticRegression
      from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

```
model = dataFrame.select('features','phish')
training, test = model_df.randomSplit([0.7,0.3],seed = 135)
```

```
[165] training.groupby("phish").count().show()
```

```
+-----+-----+
|phish|count|
+-----+-----+
|    0|  722|
|    1|  691|
+-----+-----+
```

```
[166] lr = LogisticRegression(featuresCol = "features",labelCol="phish", maxIter = 10)
      lrn = lr.fit(training)
```

```
[167] predictions = lrn.transform(test)
```

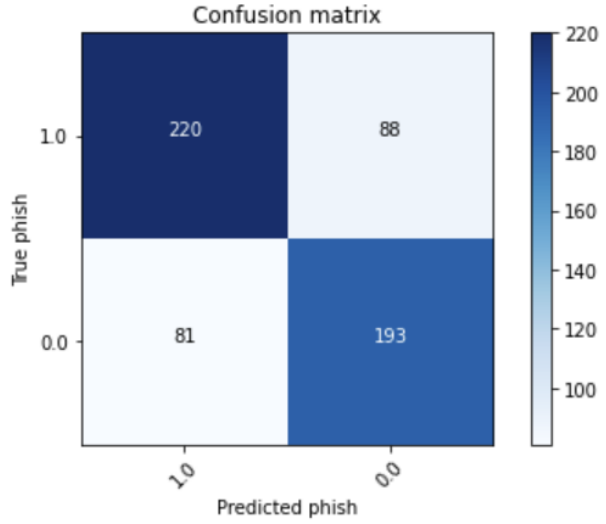
```
accuracy = predictions.filter(predictions.phish == predictions.prediction).count() / float(predictions.count())
print("Accuracy : ",accuracy)
```

```
Accuracy :  0.7096219931271478
```

Przy początkowych parametrach Accuracy osiągnęło w przybliżeniu wartość 0.71

Poniżej na Confusion Matrix, widać jak dane ze zbioru testowego zostały sklasyfikowane

```
Confusion matrix, without normalization
[[220  88]
 [ 81 193]]
```



Następnie próbowaliśmy ulepszyć model korzystając znowu z metod dostępnych w PySpark

```
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

paramGrid = (ParamGridBuilder()
             .addGrid(lr.regParam, [0.01, 0.5, 2.0])
             .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0])
             .addGrid(lr.maxIter, [1, 5, 10])
             .build())
cv = CrossValidator(estimator = lr, estimatorParamMaps = paramGrid, evaluator = eval, numFolds = 5)

cvModel = cv.fit(training)
```

Model sam wybrał dla których podanych parametrów uzyskuje najlepsze wyniki. W wyniku ulepszenia modelu wartość Accuracy zwiększyła się do 0.85

```
predictions = cvModel.transform(test)
eval.evaluate(predictions)
```

```
0.8506493506493507
```

Wybrane parametry:

```
best_model = cvModel.bestModel
print('Best (regParam): ', best_model._java_obj.getRegParam())
print('Best (maxIter): ',best_model._java_obj.getMaxIter())
print('Best (elasticNetParam): ',best_model._java_obj.getElasticNetParam())
```

```
Best (regParam):  0.01
Best (maxIter):  10
Best (elasticNetParam):  1.0
```

```
from pyspark.sql import SQLContext
sc=spark.sparkContext
sqlContext=SQLContext(sc)
weights = cvModel.bestModel.coefficients
weights = [(float(w),) for w in weights]
weightsDF = sqlContext.createDataFrame(weights, ["Feature Weight"])
weightsDF.toPandas().head(10)
```

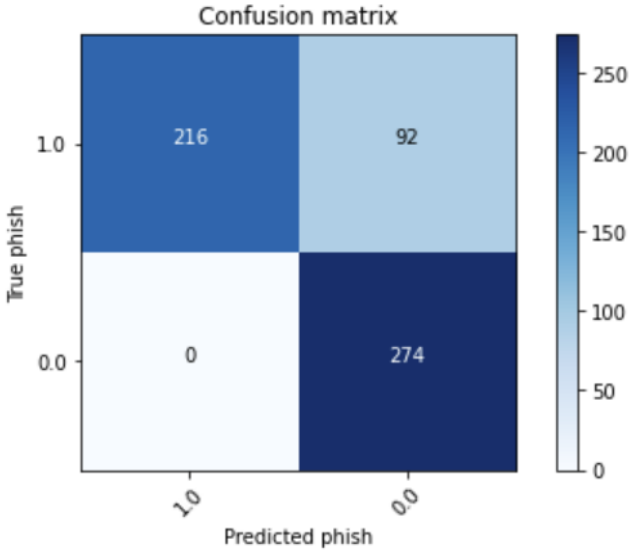
```
/content/spark-3.3.1-bin-hadoop3/python/pyspark/sql/context.py:112: FutureWarn
  warnings.warn(
```

	Feature Weight
0	-4.039788
1	0.086582
2	0.006086
3	-1.009938
4	4.032224
5	-0.002260

Patrząc na określone przez best_model wagi dla danych modelu widać, że dana określająca czy adres zawiera https jest kompletnie niepotrzebna, a wręcz przeszkadza w predykcji. Tak samo dane przechowująca informację o liczbie kropek i długości adresu. Najbardziej pomocna okazuje się informacja o tym, czy adres url zawiera ‘php’.

Dla tego Best_model Confusion Matrix wygląda następująco:

```
Confusion matrix, without normalization
[[216  92]
 [  0 274]]
```



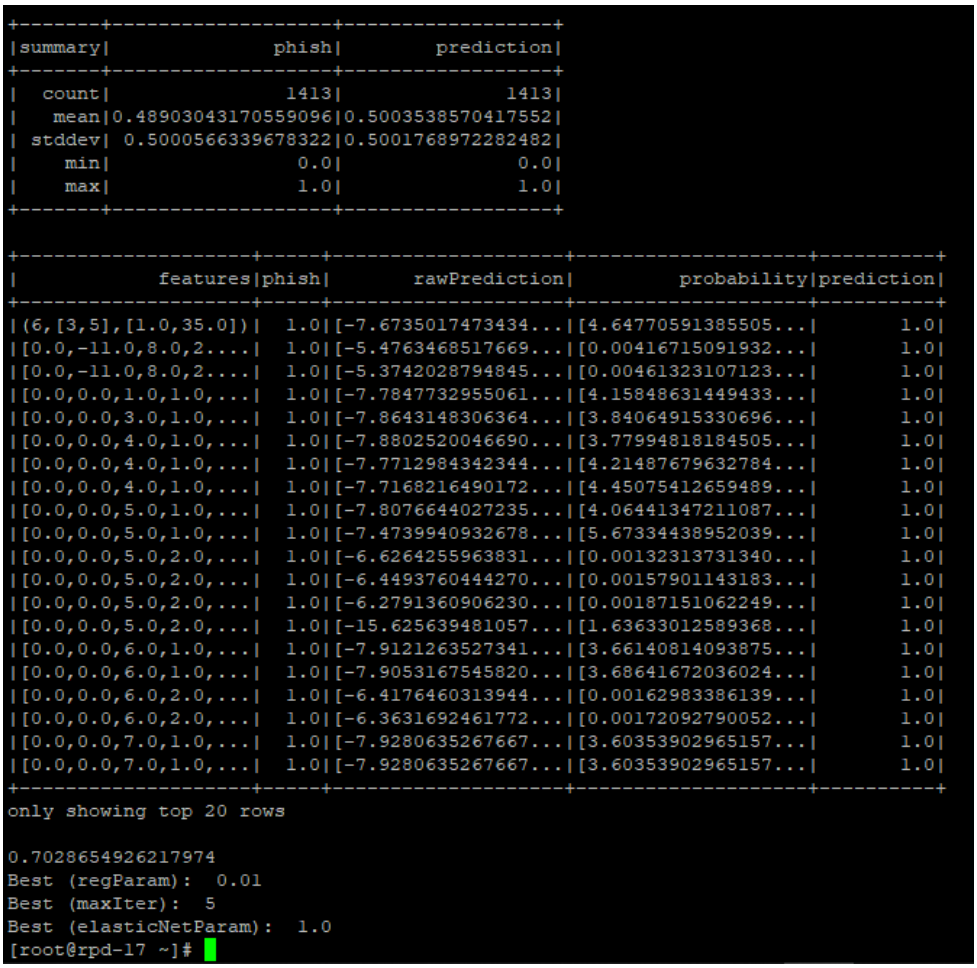
Widać, że żaden bezpieczny adres url nie został sklasyfikowany jako niebezpieczny. W praktyce zależałoby nam na odwrotnym wypadku – czyli możemy sobie pozwolić na błędne sklasyfikowanie bezpiecznych adresów jako niebezpieczne, jednak wszystkie niebezpieczne adresu url byłyby wychwycone.

Model został umieszczony na maszynie z zainstalowanym Apache Spark – 10.7.38.180.

Wyniki modelu są bardzo podobne do tych otrzymanych wcześniej:

```
[root@rpd-17 ~]# python3 model.py
23/01/24 22:20:08 WARN Utils: Your hostname, rpd-17 resolves to a loopback address: 127.0.1.1; using 10.7.38.180 instead (on inter
23/01/24 22:20:08 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/01/24 22:20:09 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes wher
+-----+-----+
|phish|count|
+-----+-----+
|    0|  722|
|    1|  691|
+-----+-----+

23/01/24 22:20:19 WARN InstanceBuilder$NativeBLAS: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
23/01/24 22:20:19 WARN InstanceBuilder$NativeBLAS: Failed to load implementation from:dev.ludovic.netlib.blas.ForeignLinkerBLAS
Accuracy :  0.7096219931271478
Training set areaUnderROC: 0.8375993682125948
+--------------------+-----+--------------------+--------------------+----------+
|            features|phish|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|(6,[3,5],[1.0,35.0])|  1.0|[-7.6735017473434...|[4.64770591385505...|       1.0|
|[0.0,-11.0,8.0,2....|  1.0|[-5.4763468517669...|[0.00416715091932...|       1.0|
|[0.0,-11.0,8.0,2....|  1.0|[-5.3742028794845...|[0.00461323107123...|       1.0|
|[0.0,0.0,1.0,1.0,...|  1.0|[-7.7847732955061...|[4.15848631449433...|       1.0|
|[0.0,0.0,3.0,1.0,...|  1.0|[-7.8643148306364...|[3.84064915330696...|       1.0|
|[0.0,0.0,4.0,1.0,...|  1.0|[-7.8802520046690...|[3.77994818184505...|       1.0|
|[0.0,0.0,4.0,1.0,...|  1.0|[-7.7712984342344...|[4.21487679632784...|       1.0|
|[0.0,0.0,4.0,1.0,...|  1.0|[-7.7168216490172...|[4.45075412659489...|       1.0|
|[0.0,0.0,5.0,1.0,...|  1.0|[-7.8076644027235...|[4.06441347211087...|       1.0|
|[0.0,0.0,5.0,1.0,...|  1.0|[-7.4739940932678...|[5.67334438952039...|       1.0|
|[0.0,0.0,5.0,2.0,...|  1.0|[-6.6264255963831...|[0.00132313731340...|       1.0|
|[0.0,0.0,5.0,2.0,...|  1.0|[-6.4493760444270...|[0.00157901143183...|       1.0|
|[0.0,0.0,5.0,2.0,...|  1.0|[-6.2791360906230...|[0.00187151062249...|       1.0|
|[0.0,0.0,5.0,2.0,...|  1.0|[-15.625639481057...|[1.63633012589368...|       1.0|
|[0.0,0.0,6.0,1.0,...|  1.0|[-7.9121263527341...|[3.66140814093875...|       1.0|
|[0.0,0.0,6.0,1.0,...|  1.0|[-7.9053167545820...|[3.68641672036024...|       1.0|
|[0.0,0.0,6.0,2.0,...|  1.0|[-6.4176460313944...|[0.00162983386139...|       1.0|
|[0.0,0.0,6.0,2.0,...|  1.0|[-6.3631692461772...|[0.00172092790052...|       1.0|
|[0.0,0.0,7.0,1.0,...|  1.0|[-7.9280635267667...|[3.60353902965157...|       1.0|
|[0.0,0.0,7.0,1.0,...|  1.0|[-7.9280635267667...|[3.60353902965157...|       1.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 20 rows
```



```
+-------+-------------------+-------------------+
|summary|              phish|         prediction|
+-------+-------------------+-------------------+
|  count|               1413|               1413|
|   mean|0.48903043170559096|0.5003538570417552|
| stddev| 0.5000566339678322|0.5001768972282482|
|    min|                0.0|                0.0|
|    max|                1.0|                1.0|
+-------+-------------------+-------------------+

+--------------------+-----+--------------------+--------------------+----------+
|            features|phish|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|(6,[3,5],[1.0,35.0])|  1.0|[-7.6735017473434...|[4.64770591385505...|       1.0|
|[0.0,-11.0,8.0,2....|  1.0|[-5.4763468517669...|[0.00416715091932...|       1.0|
|[0.0,-11.0,8.0,2....|  1.0|[-5.3742028794845...|[0.00461323107123...|       1.0|
|[0.0,0.0,1.0,1.0,...|  1.0|[-7.7847732955061...|[4.15848631449433...|       1.0|
|[0.0,0.0,3.0,1.0,...|  1.0|[-7.8643148306364...|[3.84064915330696...|       1.0|
|[0.0,0.0,4.0,1.0,...|  1.0|[-7.8802520046690...|[3.77994818184505...|       1.0|
|[0.0,0.0,4.0,1.0,...|  1.0|[-7.7712984342344...|[4.21487679632784...|       1.0|
|[0.0,0.0,4.0,1.0,...|  1.0|[-7.7168216490172...|[4.45075412659489...|       1.0|
|[0.0,0.0,5.0,1.0,...|  1.0|[-7.8076644027235...|[4.06441347211087...|       1.0|
|[0.0,0.0,5.0,1.0,...|  1.0|[-7.4739940932678...|[5.67334438952039...|       1.0|
|[0.0,0.0,5.0,2.0,...|  1.0|[-6.6264255963831...|[0.00132313731340...|       1.0|
|[0.0,0.0,5.0,2.0,...|  1.0|[-6.4493760444270...|[0.00157901143183...|       1.0|
|[0.0,0.0,5.0,2.0,...|  1.0|[-6.2791360906230...|[0.00187151062249...|       1.0|
|[0.0,0.0,5.0,2.0,...|  1.0|[-15.625639481057...|[1.63633012589368...|       1.0|
|[0.0,0.0,6.0,1.0,...|  1.0|[-7.9121263527341...|[3.66140814093875...|       1.0|
|[0.0,0.0,6.0,1.0,...|  1.0|[-7.9053167545820...|[3.68641672036024...|       1.0|
|[0.0,0.0,6.0,2.0,...|  1.0|[-6.4176460313944...|[0.00162983386139...|       1.0|
|[0.0,0.0,6.0,2.0,...|  1.0|[-6.3631692461772...|[0.00172092790052...|       1.0|
|[0.0,0.0,7.0,1.0,...|  1.0|[-7.9280635267667...|[3.60353902965157...|       1.0|
|[0.0,0.0,7.0,1.0,...|  1.0|[-7.9280635267667...|[3.60353902965157...|       1.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 20 rows

0.7028654926217974
Best (regParam):  0.01
Best (maxIter):  5
Best (elasticNetParam):  1.0
[root@rpd-17 ~]#
```

Aby ukazać jego działanie, zapisany model ML przetwarza dane odbierane ponownie z bazy Elastic postawionej na maszynie 10.7.38.174 – PhishTank i predykuje czy mogłyby być one uznane jako niebezpieczne.

```
[root@rpd-17 ~]# python3 Spark_m2.py
23/01/24 22:28:48 WARN Utils: Your hostname, rpd-17 resolves to a loopback address: 127.0.1.1; using 10.7.38.180 instead
23/01/24 22:28:48 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/01/24 22:28:48 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cla
url:http://node-sc.com, predicition: [Row(prediction=1.0)]
url:https://blitmap.rhnfts.com/, predicition: [Row(prediction=0.0)]
url:https://exosama.rfnfts.com/, predicition: [Row(prediction=0.0)]
url:https://wwverify.start.page/, predicition: [Row(prediction=0.0)]
url:http://rdfkhbpejp.duckdns.org, predicition: [Row(prediction=1.0)]
url:https://worldcup-bordape.com/, predicition: [Row(prediction=1.0)]
url:https://boredape-fifa.com/, predicition: [Row(prediction=1.0)]
url:https://micadbrazzz.blogspot.com/, predicition: [Row(prediction=0.0)]
url:https://2135816316731063.ga/, predicition: [Row(prediction=1.0)]
url:http://labgrownbeasts2-free.xyz, predicition: [Row(prediction=1.0)]
url:https://luxi3.xyz/mint, predicition: [Row(prediction=1.0)]
url:https://debanc.000webhostapp.com/, predicition: [Row(prediction=0.0)]
url:http://worldwide.lpnfts.com, predicition: [Row(prediction=1.0)]
url:https://tribelio.page/communitystandardissue, predicition: [Row(prediction=1.0)]
url:http://business197487sm0q89032id03lny9restrictedpagerecovery.co.vu/getstarted.php, predicition: [Row(prediction=1.0)]
url:https://wwwdbspaylsh.top/, predicition: [Row(prediction=1.0)]
url:https://wwwdbspaylsh.in/, predicition: [Row(prediction=1.0)]
url:https://s.ezklusrzo.icu/jp.php, predicition: [Row(prediction=1.0)]
url:https://wwwdbspaylsh.co/, predicition: [Row(prediction=1.0)]
url:https://s.ezkluerzo.icu/jp.php, predicition: [Row(prediction=1.0)]
url:https://s.ezkluersa.icu/jp.php, predicition: [Row(prediction=1.0)]
url:https://s.ezkluerso.icu/jp.php, predicition: [Row(prediction=1.0)]
url:https://www.goo-co-etc.inforlink.tanyuming.com.cn/jp, predicition: [Row(prediction=0.0)]
url:https://wwwdbspaylsh.shop/, predicition: [Row(prediction=1.0)]
url:https://taqibraz.blogspot.com/, predicition: [Row(prediction=0.0)]
url:https://itamarationlinebraz.blogspot.com/, predicition: [Row(prediction=1.0)]
url:https://dopedabrazzz.blogspot.com/, predicition: [Row(prediction=1.0)]
url:http://wwobltkubs-nom.lgb.ru/, predicition: [Row(prediction=1.0)]
url:https://costumeds.blogspot.com/, predicition: [Row(prediction=0.0)]
url:https://sbi-222hgdv.web.app/, predicition: [Row(prediction=0.0)]
url:https://hyp.ae/kMqET, predicition: [Row(prediction=1.0)]
url:https://crditagricolecairegoonalc.web.app/, predicition: [Row(prediction=1.0)]
url:https://sirenita-kub.blogspot.com/, predicition: [Row(prediction=1.0)]
url:https://uiriocabrabrazzz.blogspot.com/, predicition: [Row(prediction=1.0)]
url:https://dapp.santaclub.xyz/mint-nft, predicition: [Row(prediction=0.0)]
url:https://serviceman.ch/mtb-auth/, predicition: [Row(prediction=1.0)]
url:https://149.127.241.253/, predicition: [Row(prediction=0.0)]
url:https://www.paylah.pink/, predicition: [Row(prediction=0.0)]
url:https://vinted-be.sales-3ds.info/cash10142824, predicition: [Row(prediction=0.0)]
url:https://iban-be.sales-3ds.info/cash10142824, predicition: [Row(prediction=0.0)]
url:http://usuario-ciudad.com/, predicition: [Row(prediction=1.0)]
url:https://reveal-valhalla.netlify.app/, predicition: [Row(prediction=1.0)]
url:http://wwobltkubs-neesc.lgb.ru/, predicition: [Row(prediction=1.0)]
url:https://www.direct.smbc.co.jp.vipan.net/aib/wroywzzzsoil0tyvvpx5e_jsp, predicition: [Row(prediction=0.0)]
url:https://access-rarible.net/upp/, predicition: [Row(prediction=1.0)]
url:https://community-uni.org, predicition: [Row(prediction=1.0)]
url:https://www.pousadavolola.ga/, predicition: [Row(prediction=0.0)]
url:http://wwobltkubs-nxs.lgb.ru/, predicition: [Row(prediction=1.0)]
url:https://passandbrazzz.blogspot.com/, predicition: [Row(prediction=1.0)]
url:https://etc-ppioopp.organiccrap.com/, predicition: [Row(prediction=0.0)]
```

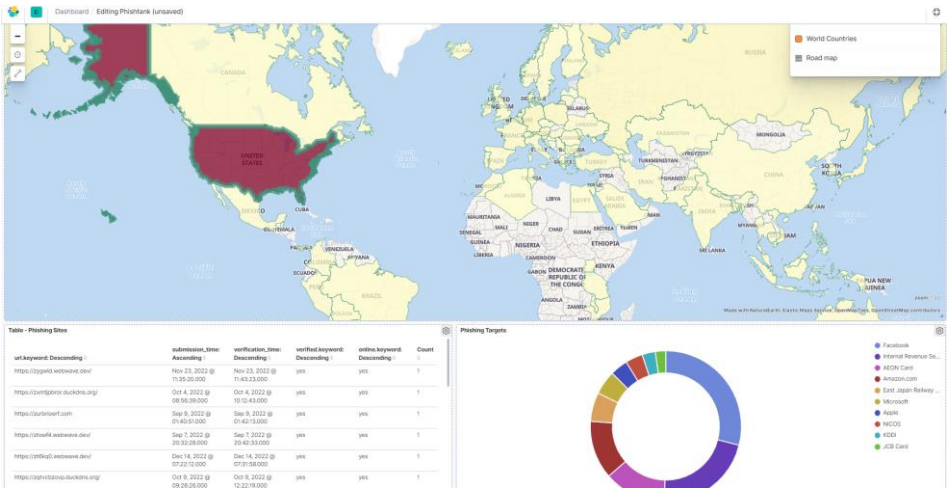
7. Zaprezentowanie wyników i wnioski końcowe

Poniżej zaprezentowano dashboardy w Kibanie:

- I Dashboard – PhishTank

Dashboard prezentuje mapkę przedstawiającą, w których krajach zarejestrowane było najwięcej domen phishingowych. Można zaobserwować, że najwięcej domen pochodziło z USA i względem innych krajów była to zdecydowana przewaga.

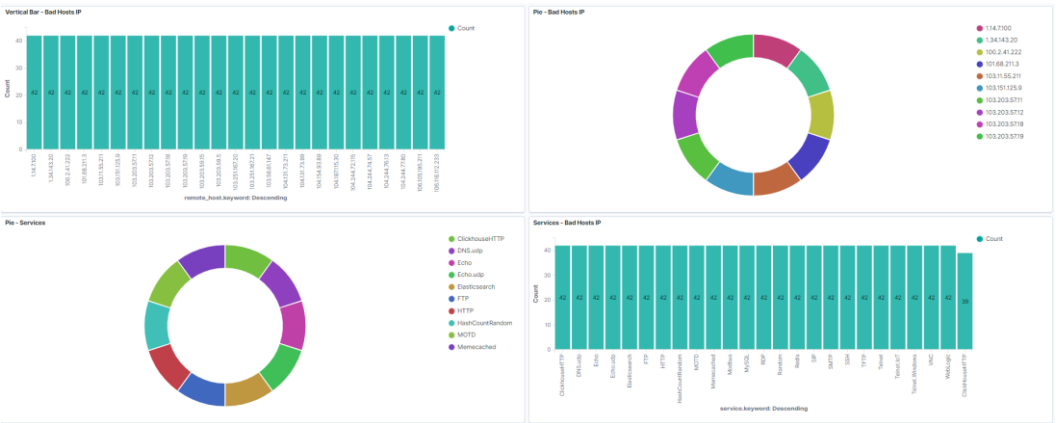
Dodatkowo, w prawym, dolnym rogu zaprezentowano wykres kołowy prezentujący, pod które domeny najczęściej podszywali się atakujący. Jak można zaobserwować najczęściej atakujący podszywali się pod Facebooka (I miejsce), Amazona (II miejsce), Apple (III miejsce), następnie na liście znalazły się między innymi takie firmy jak: Microsoft, Millenium Bank



Rysunek 10 Prezentacja dashboardu PhishTank

- II Dashobard – Bad Hosts

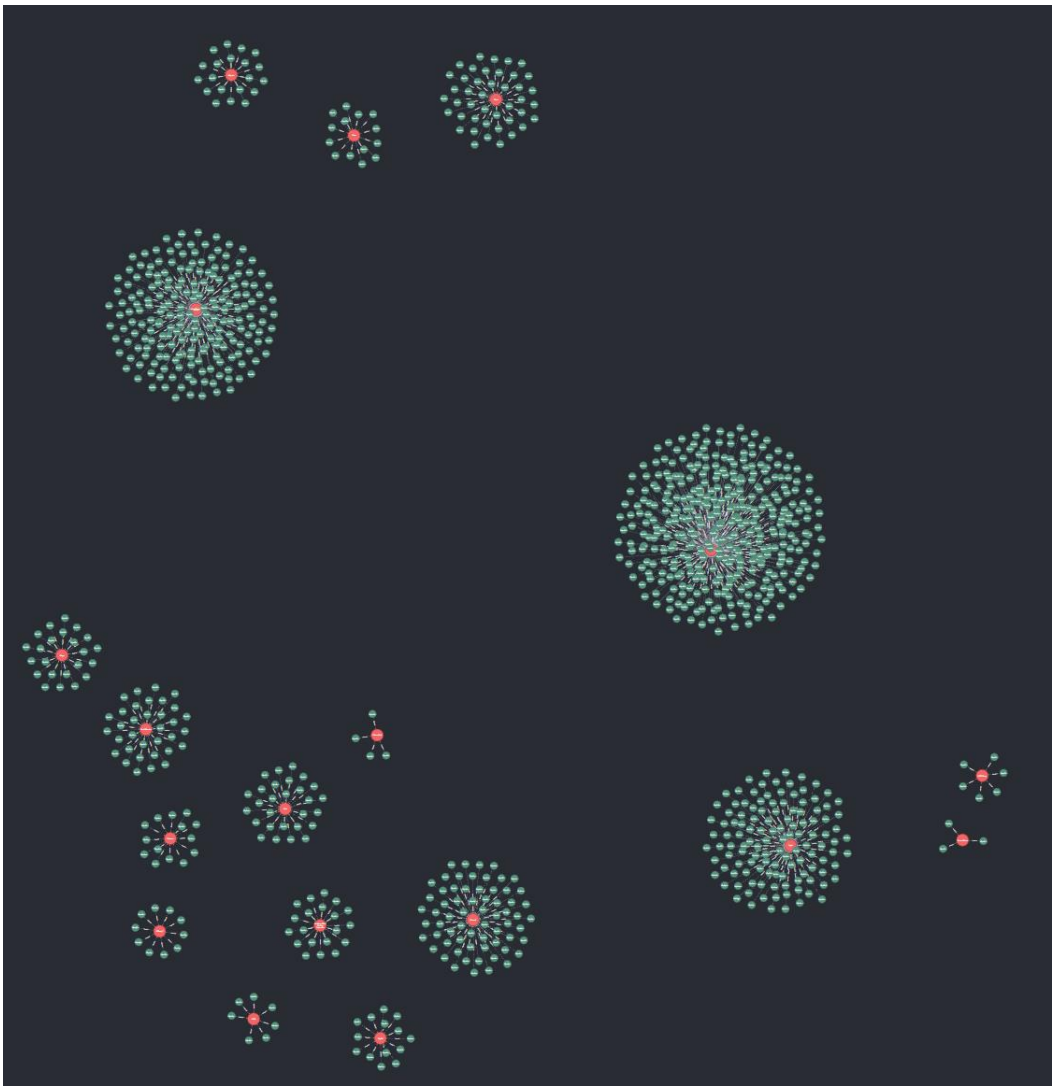
Dashboard prezentuje wykresy kołowe, górny wykres wskazuje, z których adresów IP odnotowywano ataki najczęściej (liczba dni przez które atak odnotowywano). Dolny wykres wskazuje, które usługi atakowano najczęściej.



Rysunek 11 Prezentacja dashboardu Bad Hosts

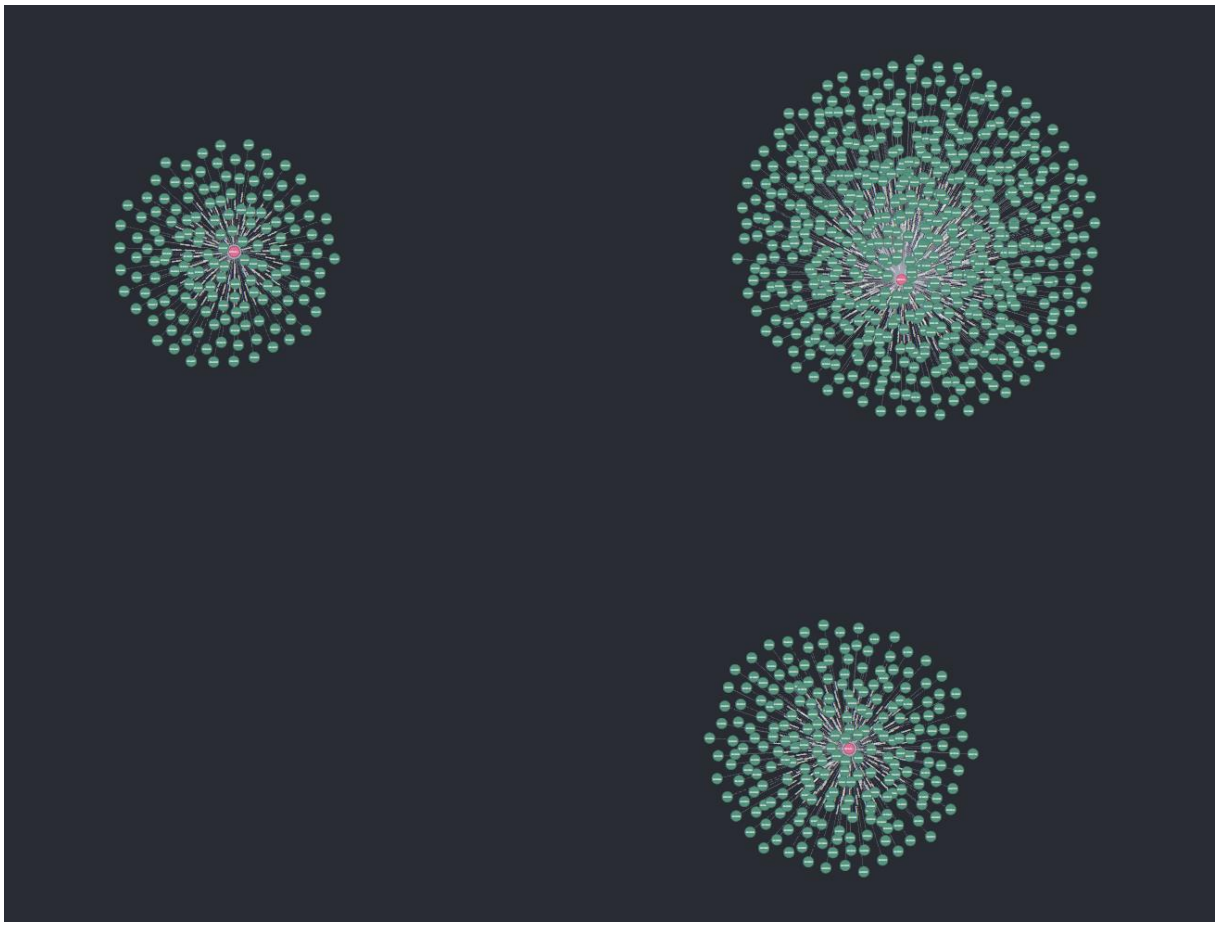
Poniżej zaprezentowano wyniki opracowania przetwarzanych danych w Neo4j.

Na pierwszym zrzucie zielone węzły oznaczają adresy URL, które podszywały się pod phishingowane cele, które zostały oznaczone kolorem czerwonym. Widać tutaj, że największa liczba ataków przypada na duże korporacje takie jak Amazon, Facebook czy Apple.



Rysunek 12 Prezentacja celów ataków phishingowych w Neo4j

Na drugim zrzucie zielone węzły oznaczają ID zgłoszenia ataku, natomiast węzły oznaczone kolorem czerwonym oznaczają datę, w której atak został zgłoszony.



Rysunek 13 Prezentacja rozkładu dni ataków phishingowych ze względu na datę w Neo4j