Security Analysis Presentation

Initial Premise

- CyberSecurity Threats Are On The Rise
  - Reported Data Breaches in 2015
    - 781
  - Estimated Global Cost of Cyber Attacks Annually
    - $400 Billion
  - Projected Global Cost of Cyber Attacks in 2019
    - $2.1 Trillion
  - Average Annual Amount of CyberSecurity Incidents
    - 80-90 million

Initial Premise

- What can we do to mitigate this situation?
  - Analysis of existing cyber attack data
    - Attempt to find patterns and/or other identifiable nuances that could possibly help us prevent attacks to valued resources
  - Updating our systems to the latest technologies
    - Keeping systems updated prevents us from being exposed to vulnerabilities that are already patched and/or mitigated. Allows us to minimize the amount of risk that our systems are vulnerable to.
  - Leveraging Machine Learning
    - Using machine learning algorithms we could possibly automate the process of Data Analysis using specific algorithms designed for identifying clusters in data. This data would serve to help us identify certain recurring themes within isolated and group attacks, possibly allowing us to shore up our defenses when and where necessary.

Data

- Real Production Environment
- Real Attacks Received From External Forces
  - Location: Marist College
  - System Attacked: Marist Mainframes located in Donnelly
  - Logging System: BlackRidge 1Gb Authentication Gateway
  - GeoLocation-ISP: geo_cheese v. 1.0 Python Module
- Data Collected
  - SRC_IP: What IP address did the attack originated from
  - SRC_Port: What port was the attack sent out of
  - DEST_IP: What IP address was targeted
  - DEST_Port: Which port was targeted
  - City: The city of the Attacking IP Address
  - Host: Host address of the attacking IP's ISP
  - Subdivision: The principality of the attacking IP Address
  - Name: The name of the Attacking IP's ISP
  - Host_IP: The Attacking IP's ISP IP
  - Lat/Long: The latitude and longitude associated with the Attacking IP
  - Country: The country of the Attacking IP
  - Postal: Postal Code (if applicable) of the Attacking IP
  - ASN: Autonomous System Number of ISP

GeoLocation-ISP


```python
import xml
import backup_query as backup
import reverse_latlng_google as reverse_g
import random_ip
import geoip2.database
import geocoder
import re
import json
import urllib2
import sys
import os.path
import update_db
from unidecode import unidecode
from BeautifulSoup import BeautifulSoup

# Check if database exists, if not download database from MaxMind
def check_db():
    if os.path.isfile("./GeoLite2-City.mmdb"):
        return True
    else:
        try:
            print("GeoLite2-City.mmdb is missing, attempting to acquire file...\n")
            update_db.get_geo_file()
            check_db()
        except:
            error = sys.exc_info()[0]
            print("Error: " + str(error))
            print("Failed to acquire GeoLite2-City.mmdb...\n")

# For filtering string of unicode, and prepare for printing
def to_string(word):
    if isinstance(word, unicode) and word:
        return str(unidecode(word))
    else:
        return str(word)

# For removing HTML Tags for web scrape
def remove_tags(text):
    return ''.join(xml.etree.ElementTree.fromstring(text).itertext())

# Main function to retrieve GeoISP data about target IP address
def find_loc(mmdb_file, my_ip):
```

Data

{"src": "118.71.56.33", "src_port": "63824", "dest": "148.100.49.187", "dest_port": "23", "city": "Hanoi", "host": "ip-address-pool-xxx.fpt.vn", "subdivision": "Thanh Pho Ha Noi", "name": "FPT Telecom Company", "ip": "118.71.56.33", "lat": "21.0333", "country": "Vietnam", "postal": "0000", "ASN": "18403", "long": "105.85"}
{"src": "141.212.122.127", "src_port": "49896", "dest": "148.100.49.186", "dest_port": "102", "city": "Ann Arbor", "host": "researchscan382.eecs.umich.edu", "subdivision": "Michigan", "name": "University of Michigan", "ip": "141.212.122.127", "lat": "42.2776", "country": "United States", "postal": "48109", "ASN": "36375", "long": "-83.7409"}
{"src": "182.186.244.72", "src_port": "47447", "dest": "148.100.49.187", "dest_port": "23", "city": "Faisalabad", "host": "182.186.244.72", "subdivision": "Punjab", "name": "Pakistan Telecommuication company limited", "ip": "182.186.244.72", "lat": "31.4167", "country": "Pakistan", "postal": "38000", "ASN": "45595", "long": "73.0833"}
{"src": "1.9.157.132", "src_port": "11155", "dest": "148.100.49.187", "dest_port": "23", "city": "Kajang", "host": "1.9.157.132", "subdivision": "Selangor", "name": "TM Net", "ip": "1.9.157.132", "lat": "3.0094", "country": "Malaysia", "postal": "43000", "ASN": "4788", "long": "101.7755"}
{"src": "84.90.65.59", "src_port": "51323", "dest": "148.100.49.187", "dest_port": "23", "city": "Seixal", "host": "pa13-84-90-65-59.netvisao.pt", "subdivision": "Lisbon", "name": "Cabovisao, televisao por cabovisao, sa", "ip": "84.90.65.59", "lat": "38.9259", "country": "Portugal", "postal": "2705-740", "ASN": "13156", "long": "-9.3938"}
{"src": "14.160.18.65", "src_port": "54278", "dest": "148.100.49.186", "dest_port": "23", "city": "Hanoi", "host": "static.vnpt.vn", "subdivision": "Thanh Pho Ha Noi", "name": "VDC", "ip": "14.160.18.65", "lat": "21.0333", "country": "Vietnam", "postal": "0000", "ASN": "45899", "long": "105.85"}
{"src": "117.196.159.157", "src_port": "52605", "dest": "148.100.49.187", "dest_port": "23", "city": "Ernakulam", "host": "117.196.159.157", "subdivision": "Kerala", "name": "BSNL", "ip": "117.196.159.157", "lat": "9.9833", "country": "India", "postal": "682011", "ASN": "9829", "long": "76.2833"}
{"src": "117.196.159.157", "src_port": "52605", "dest": "148.100.49.187", "dest_port": "23", "city": "Ernakulam", "host": "117.196.159.157", "subdivision": "Kerala", "name": "BSNL", "ip": "117.196.159.157", "lat": "9.9833", "country": "India", "postal": "682011", "ASN": "9829", "long": "76.2833"}
{"src": "87.227.225.43", "src_port": "43594", "dest": "148.100.49.187", "dest_port": "23", "city": "Elena", "host": "87.227.225.43", "subdivision": "Oblast Veliko Tarnovo", "name": "Mobiltel Ead", "ip": "87.227.225.43", "lat": "42.9333", "country": "Bulgaria", "postal": "5070", "ASN": "8717", "long": "25.8833"}
{"src": "59.99.212.124", "src_port": "30528", "dest": "148.100.49.186", "dest_port": "23", "city": "Gramam", "host": "59.99.212.124", "subdivision": "Kerala", "name": "BSNL", "ip": "59.99.212.124", "lat": "10.7209", "country": "India", "postal": "679579", "ASN": "9829", "long": "75.946"}
{"src": "117.217.210.14", "src_port": "62016", "dest": "148.100.49.186", "dest_port": "23", "city": "Nagercoil", "host": "117.217.210.14", "subdivision": "Tamil Nadu", "name": "BSNL", "ip": "117.217.210.14", "lat": "8.1939", "country": "India", "postal": "629001", "ASN": "9829", "long": "77.4314"}
{"src": "122.54.215.10", "src_port": "60088", "dest": "148.100.49.187", "dest_port": "23", "city": "Manila", "host": "122.54.215.10.pldt.net", "subdivision": "Metro Manila", "name": "Philippine Long Distance Telephone", "ip": "122.54.215.10", "lat": "14.5955", "country": "Philippines", "postal": "1010", "ASN": "9299", "long": "120.9721"}
{"src": "122.54.215.10", "src_port": "60088", "dest": "148.100.49.187", "dest_port": "23", "city": "Manila", "host": "122.54.215.10.pldt.net", "subdivision": "Metro Manila", "name": "Philippine Long Distance Telephone", "ip": "122.54.215.10", "lat": "14.5955", "country": "Philippines", "postal": "1010", "ASN": "9299", "long": "120.9721"}
{"src": "190.249.151.116", "src_port": "12476", "dest": "148.100.49.186", "dest_port": "23", "city": "Medellin", "host": "cable190-249-151-116.epm.net.co", "subdivision": "Antioquia", "name": "UNE", "ip": "190.249.151.116", "lat": "6.2518", "country": "Colombia", "postal": "050012", "ASN": "13489", "long": "-75.5636"}

K-Means.py

- k-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

- The problem is computationally difficult (NP-hard); however, there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum. These are usually similar to the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both algorithms. Additionally, they both use cluster centers to model the data; however, k-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

- The algorithm has a loose relationship to the k-nearest neighbor classifier, a popular machine learning technique for classification that is often confused with k-means because of the k in the name. One can apply the 1-nearest neighbor classifier on the cluster centers obtained by k-means to classify new data into the existing clusters. This is known as nearest centroid classifier or Rocchio algorithm

K-Means.py

- Given a set of observations ($\mathbf{x}_1$, $\mathbf{x}_2$, …, $\mathbf{x}_n$), where each observation is a *d*-dimensional real vector, *k*-means clustering aims to partition the *n* observations into *k* (≤ *n*) sets $\mathbf{S} = \{S_1, S_2, \ldots, S_k\}$ so as to minimize the within-cluster sum of squares (WCSS) (sum of distance functions of each point in the cluster to the K center). In other words, its objective is to find:

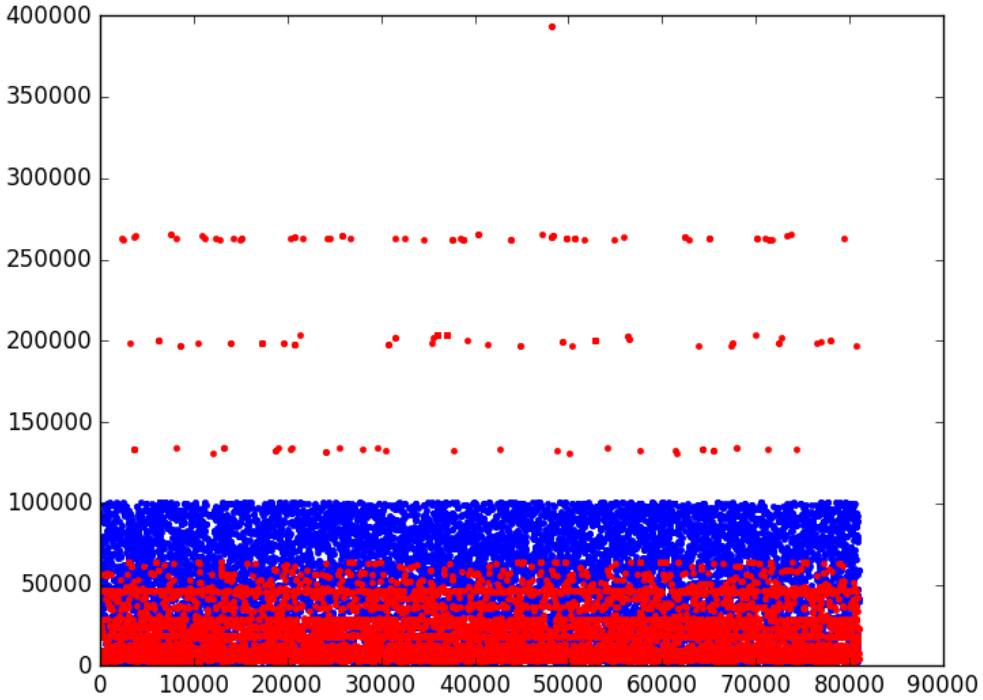
$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

- where $\boldsymbol{\mu}_i$ is the mean of points in S_i.

K-Means.py

```
"""
================================
KMeans
================================
Modified in class by Dr. Rivas
"""

import ...

# How many times to run KMeans.py
run_times = 1

for i in range(run_times):
    # read digits data & split it into X (training input) and y (target output)
    X = genfromtxt('Dataset/IP-Port.csv', delimiter=' ')

    plt.plot(X[:, 0], X[:, 1], '.')
    plt.plot(X[:, 0], X[:, 2], 'r.')
    # plt.show()

    bestk = []
    kc = 0
    result_num = len(os.listdir('KMeans_Results'))
    with open('KMeans_Results/kmeans_results' + str(result_num) + '.txt', 'w') as file:
        for clusters in range(2, 101, 1):
            kf = KFold(n_splits=10)
            # clusters = 85
            kscore = []
            k = 0

            print("============ KScore ============")
            file.writelines("============ KScore ============" + "\n")
            for train, test in kf.split(X):
                #print("%s %s" % (train, test))
                X_train, X_test = X[train], X[test]

                #time.sleep(100)

                # we create an instance of Neighbors Classifier and fit the data.
                clf = KMeans(n_clusters=clusters)
                clf.fit(X_train)

                labels = clf.labels_
                kscore.append(metrics.silhouette_score(X_train, labels, metric='euclidean'))

                print(kscore[k])
                file.writelines(str(kscore[k]) + "\n")
                k=k+1
            print("============ Clusters ============")
            file.writelines("============ Clusters ============" + "\n")
            print (clusters)
            file.writelines(str(clusters) + "\n")
            bestk.append(sum(kscore)/len(kscore))
            print("============ BestK[KC] ============")
            print(bestk[kc])
            file.writelines("============ BestK[KC] ============" + "\n")
            file.writelines(str(bestk[kc]) + "\n")
            kc+=1

            # to do here: given this array of E_outs in CV, find the max, its
            # corresponding index, and its corresponding value of clusters
            print("============ BestK Array ============")
            file.writelines("============ BestK Array ============" + "\n")
            print(bestk)
            file.writelines(str(bestk) + "\n")

print("============ BestK ============")
file.writelines("============ BestK ============" + "\n")
print(max(bestk))
file.writelines(str(max(bestk)))
```

Graph of Data



K-Means Results

- ========== BestK Array ==========
- [0.32275529884040943, 0.3476449452820386, 0.33865681258417046, 0.34637379668274176, 0.32932986508833562, 0.32832920977268981, 0.31320283348200539, 0.31886748405655435, 0.31820892144513224, 0.31845371261665029, 0.32046593104135479, 0.32286548256595637, 0.32056327021535508, 0.31722276199635341, 0.31584830049232065, 0.31188241179739828, 0.31112891627456252, 0.30947299719065718, 0.30760707749542526, 0.3084086056466363, 0.30368203309162312, 0.30508395648202868, 0.30252013717324239, 0.30046506435532061, 0.30177186741724688, 0.30171240375402286, 0.29830348375265492, 0.30183261163338165, 0.30283258149465986, 0.30404816961996295, 0.30261316683214728, 0.30453145554052014, 0.30538318604391407, 0.30391821459526691, 0.30582448451648764, 0.30637133408877071, 0.3032977782033654, 0.30487187395570231, 0.30435387759016252, 0.3042724634594095, 0.30350387339127938, 0.30495206161168464, 0.30436961450241745, 0.30350062274920558, 0.30461153064788088, 0.30434630177151117, 0.30506188045178184, 0.30494880108241496, 0.30258117377571586, 0.30418233062111366, 0.30122629571454979, 0.30251978749831865, 0.30319461501828565, 0.30495433069524458, 0.3035787223820311, 0.30414482543574606, 0.30381740183736239, 0.3028383016343934, 0.30340374154819499, 0.30305707503319124, 0.30295986512645334, 0.30081422083849496, 0.30234574575792245, 0.30096957882411346, 0.30328101374168165, 0.30522020348313961, 0.30102153923611474, 0.30261311089488058, 0.30266999369057207, 0.30379765727671815, 0.30333409181821602, 0.30413308936767064, 0.30316524204455975, 0.30449222953493049, 0.30368704555489495, 0.30334009880590401, 0.30234806390945862, 0.30460630148040513, 0.30226308569850124, 0.30463881099772944, 0.30613710647376197, 0.30527782926172753, 0.303974427245221, 0.30417299145636029, 0.30542883076145538, 0.30501248019413707, 0.3035779083918414, 0.30521295645669044, 0.30405001953743016, 0.30694141289619126, 0.30362681004183528, 0.30405982631441669, 0.30455052007273747, 0.30521351727692236, 0.30548936544538952, 0.30550292450903982, 0.30481011218268017, 0.30583745561796144, 0.30620261289739414]
- **========== BestK ==========**
- **0.347644945282**

Moving Forward

- Apply more datasets to be analyzed by K-Means.py
- Spectral Analysis
  - In multivariate statistics and the clustering of data, spectral clustering techniques make use of the spectrum (eigenvalues) of the similarity matrix of the data to perform dimensionality reduction before clustering in fewer dimensions. The similarity matrix is provided as an input and consists of a quantitative assessment of the relative similarity of each pair of points in the dataset.
  - In application to image segmentation, spectral clustering is known as segmentation-based object categorization.
- More Perspective
- More Real World Application

Spectral Clustering

- Traceback (most recent call last):
- File "C:/Users/Tien/Documents/GitHub/CMPT404-ARTIFICAL-INTELLIGENCE/Liengtiraphan-Project/Project Resources/Spectral_Clustering.py", line 50, in <module>
- clf.fit(X_train)
- File "C:\Users\Tien\Anaconda3\lib\site-packages\sklearn\cluster\spectral.py", line 463, in fit
- assign_labels=self.assign_labels)
- File "C:\Users\Tien\Anaconda3\lib\site-packages\sklearn\cluster\spectral.py", line 258, in spectral_clustering
- eigen_tol=eigen_tol, drop_first=False)
- File "C:\Users\Tien\Anaconda3\lib\site-packages\sklearn\manifold\spectral_embedding_.py", line 312, in spectral_embedding
- largest=False, maxiter=2000)
- File "C:\Users\Tien\Anaconda3\lib\site-packages\scipy\sparse\linalg\eigen\lobpcg\lobpcg.py", line 455, in lobpcg
- activeBlockVectorBP, retInvR=True)
- File "C:\Users\Tien\Anaconda3\lib\site-packages\scipy\sparse\linalg\eigen\lobpcg\lobpcg.py", line 101, in _b_orthonormalize
- gramVBV = cholesky(gramVBV)
- File "C:\Users\Tien\Anaconda3\lib\site-packages\scipy\linalg\decomp_cholesky.py", line 81, in cholesky
- check_finite=check_finite)
- File "C:\Users\Tien\Anaconda3\lib\site-packages\scipy\linalg\decomp_cholesky.py", line 30, in _cholesky
- raise LinAlgError("%d-th leading minor not positive definite" % info)
- numpy.linalg.linalg.LinAlgError: 6-th leading minor not positive definite