Clustering based Intrusion Detection System

A Report Submitted in partial fulfillment of course CS G513

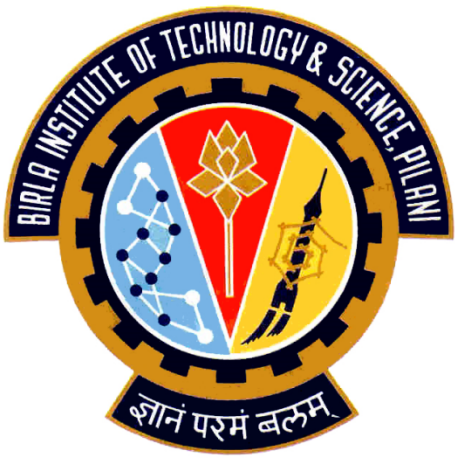
Network Security Project Report

Aditya Raj Agarwal
2015A7PS0093P

Mrigesh Madaan
2015A7PS0146P

Project Instructor
Ashutosh Bhatia
Assistant Professor

Computer Science & Information Systems Group
Birla Institute of Technology and Science, Pilani



Birla Institute of Technology & Science,

Pilani, Rajasthan 333031

Index

Index **2**

Abstract **3**

1. Introduction **4**

2. Background **5**

Denial of Service (DoS) Attacks 5

Types of Anomalies 5

Clustering 6

K-means Clustering 6

3. Preparatory Work **7**

Dataset **7**

X-means 8

4. Collective Anomaly Detection **12**

Algorithm 12

Implementation 13

5. Result **14**

6. Suggested Improvements **17**

Problem with sorting clusters based on size: 17

Dealing with Distributed DoS attacks: 17

7. Code Snippets **19**

1. X-means 19

2. CAD 21

8. References **23**

Abstract

The field of cyber security is based on the three major pillars of Confidentiality, Integrity, and Availability. In this project we have looked into the subdivision of Availability, and in particular the Denial of Service (DoS) attacks, via anomaly detection. The process is carried out by creating clusters of network usage based on certain attributes, using X-means clustering. The results obtained after further calculations on the clusters are noted and verified against benchmark Intrusion Detection DARPA dataset by MIT Lincoln Laboratory.

1. Introduction

Collective anomaly detection in a network flow is one of the widely recognised DoS attack detection methods. In this project we look at the work of Ahmed and Mehmood on Network traffic analysis based on collective anomaly detection[1]. We implement the said topic in python, using the standardised DARPA dataset[2] of MIT Lincoln labs[3].

Before delving into the main algorithm, we will start with talking about the various concepts used all through the paper, in section 2. This includes, a discussion on what are DoS attacks, what are the different types of anomalies in a dataset, what is clustering. The section closes with one of the standard clustering algorithms, the K-means algorithm.

The next sections talks about the implementation of the background work put in from our side. It starts with the discussion of converting the DARPA dataset into a usable format, and then about the implementation of the X-means algorithm. Although these details could be mentioned under the previous section, doing so will not be doing justice to the effort put in the two things as they become the base of the Collective Anomaly Detection algorithm.

In section 4 we discuss the intricacies of the CAD algorithm in detail. This section is divided in two parts, the first being the algorithm which we are using. A part of it is taken from the paper by Ahmed and Mehmood, whereas here itself we define our modifications, corrections, and additions to the algorithm. One of the additions being the detection of SMURF attacks. The second part of this section is an instruction on how can anyone reading this report see the algorithm in action. The complete source code is provided on this github[4] repository.

After discussing the CAD algorithm and our implementation we talk about the results obtained while testing against 100,000 records from the dataset. The results obtained are very satisfactory. Here we have detected two kinds of DoS attacks, Back and Neptune. The addition to the previous work has been the detection of the Smurf attacks, which fall under the category of Distributed Denial of Service attacks. The accuracy of the algorithm against the standards provided with the dataset has been given in the closing of the section.

At the end we have provided two of the very pivotal codes for the functions of X-Means clustering and calculating the Collective Anomalies. Although the whole code is provided in the repository linked above, these two pieces of code are referenced at several places across the paper and hence it seems apt to add them in the paper.

2. Background

Denial of Service (DoS) Attacks

DoS, or the Denial of Service attack, in simple terms is any attack which stops legitimate users from accessing resources there are authorised to access. Wikipedia[1] defines DoS attacks as, “a denial-of-service attack (DoS attack) is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet”. Achieving DoS attack requires an adversary to send too many superfluous requests, also known as flooding, so that when any actual request is encountered, the host system (server) does not have the adequate resources to cater to that. A simple example of a DoS attack is when the access to a particular website is hampered by an adversary who continuously sends multiple access requests to the server maintaining the website, hence keeping the site busy to cater to any other user. Here it should also be noted that performing DoS attack does not require any prior permission, as anyone can access a public website, and hence the attacks are considered tough to deal with. The type of DoS attack we deal here is a network flow attack where an adversary machine sends multiple heavy requests to the victim machine.

Types of Anomalies

Anomalies are relatively small patterns in the data set which show behaviour not in correspondence with a well defined characteristic of the overall pattern. This anomalous behaviour results in the following three kinds of observable anomalies.

1. **Point Anomaly**: When an individual instance of data shows a particular variation different from the general pattern. For example, a single request from an IP like 001.X.Y.Z, when all other requesting IPs belong to subnet 120.150.0.0/16.

2. **Contextual Anomaly**: When a data instance or a group of data instances is anomalous in a particular context, but behaves like the general set otherwise. For example, multiple requests are obtained from the subnet 001.002.0.0/16 which show similar payload value patterns as shown by the general data set which comprises mainly of requests from subnet 120.150.0.0/16

3. **Collective Anomaly**: When a collection of related data is altogether different than the rest of the set. For example, when a lot of requests are obtained from a particular IP, through the same protocol, of the same payload length.

Clustering

Clustering, or cluster analysis, is the method of grouping elements such that elements of the same group are similar to each other in more regards than to the members of any other group. The elements can be grouped by any attribute of choice. In our project, we have to cluster twice, first on the the four tuple of <sourceIP, destinationIP, payload, protocol> to get the first set of clusters, on which we apply clustering another time using just the payload length. It has been shown that these four-tuple attributes are sufficient to identify DoS attacks[2]. There is a good variety of clustering algorithms, of which we have applied the X-means algorithm, which is a variant of the vector quantization based K-means algorithm.

K-means Clustering

K means is a vector quantization based clustering algorithm, which aims at dividing a particular set of data elements in k number of clusters. The basic idea of the algorithm can be summarised in the following pseudocode.

Input: (i) K (ii) $x_1,...,x_j$
1. Place centroid $c_1,...,c_k$ on the data set
2. While centroid values change
3. Place each data point x_1 to x_j in the cluster of the centroid closest to itself
4. New centroid is the arithmetic mean of all the x_i values in the cluster
5. End
6. Return the set of clusters

Selecting the value of K becomes an important task here. For that very purpose, in our project we have used a variation of K-means algorithm, X-means, which will be explained in the next section.

3. Preparatory Work

Dataset

The dataset used for this project is the MIT DARPA dataset. This dataset contains packets sniffed off a real network in MIT Lincoln Lab. It also contains tcpdumps of various intrusion attacks along with the normal traffic flow in a network. It was first collected in 1998 over a period of 7 weeks. We were specifically interested in using the seventh week training dataset because it includes majority of DOS and DDOS attacks(Neptune, Back, Smurf etc).
The data was viewable only in wireshark. From wireshark, we exported the data into a .json file. This resulted in a 8GB .json file with approximately 1 Million records where each record had information of all the layers involved in the OSI model. Now, we had to extract the 4 tuple information (<sourceIP, destinationIP, Protocol, Payload-length>) from each record. Since this data was too big to parse it in random access memory, we had to design an iterative json parser tool which would read the data iteratively instead of trying to parse the data by bringing the whole data in random access memory.
After the conversion we had data in the required 4 tuple form. One of the records of the whole data is shown below (Protocol:'6' represents TCP protocol).

```
{
        sourceIP : "172.16.113.105",
        destinationIP: "194.27.251.21",
        payloadLength: "1460",
        Protocol: "6"
}
```

In this data, the source and the destination IPs are represented as a string. However, for clustering the data, we need these values to be integers. Hence, these values were further converted into integers using the following logic:

If IP address is A.B.C.D, Then the corresponding IP address in the integer form will be-

IP = A*(255^3) + B*(255^2) + C*255 + D

For example, the IP address "172.16.113.105" in its integer form will be 2853065820.

This representation is chosen, keeping in mind that IP addresses that are in the same subnet are more likely to be in close proximity of each other. The IP addresses that are the same subnet will have the first bits common among them. Hence, we are giving the highest weight(255^3) to the first 8 bits of the IP address and assigning exponentially less weights afterwards.

Finally, our the data that will be fed as an input to the clustering algorithm was a 2-dimensional array with each row representing a 4 column record. The first, second, third and fourth column represent the souceIP, destinationIP, payload length and protocol respectively.

```
[
  [3238153505, 2853065654, 24, 6],
   [2853065654, 3252374708, 0, 6],
   [3238153505, 2853065667, 0, 6],
                    .
                    .
                    .
   [2239021307, 2853065667, 0, 6],
   [3252374708, 2853066138, 0, 6],
    [3435373765, 2853066459, 0, 6]
]
```

X-means

X-means[8] is a variation of K-means algorithm which automatically decides the best value of K for clustering, while iteratively dividing and subdividing the datasets in K clusters while updating the value of K to finally reach an optimal value. The algorithm can be summarised in a seemingly simple pseudocode as follows.

1. K = 2, K_{max} = sqrt(dataset.size/2)
2. While (K<K_{max})
3. Improve-Params
4. Improve-Structure
5. Stop
6. Return the best BIC score model

The judgement of a cluster being better than another cluster is done on the basis of the BIC, or Bayesian information criterion, value. BIC of a cluster C is derived based on the following equation-

$$P(x_i) = \sum_{n=1}^{K} \underbrace{P(x_i \in D_n)}_{\substack{\text{prob. data point i is} \\ \text{an element of cluster } D_n}} \cdot \underbrace{P(x_i | x_i \in D_n)}_{\substack{\text{prob. element i is positioned} \\ \text{at } x_i \text{ if in cluster } D_n}}$$

The log-likelihood of the above equation for all the data points becomes-

$$l(D) = \log \prod_i P(x_i)$$

The maximum log likelihood of the data point is found out by calculating the partial derivative of the above equation and equating it to zero [7]. After having a method of calculating which cluster model is better, we move ahead with defining the Improve-Params and Improve-Structure from the pseudocode mentioned earlier.

Improve-Params part of the function runs the conventional K-means algorithm on the data set. A track of the best BIC value is kept and every time a new cluster is made, its BIC value is compared against the best BIC value and the latter is updated if the need be. A similar counter is kept for the best value of K as well, getting which is one of the main aims of the variation in the algorithm over the traditional k-means algorithm.

Improve-Structure part of the function decides where should the new centroids should appear in the dataset, if the maximum k value hasn't been reached yet. It takes in the clusters produced in the Improve-Params part as input. It creates two centroids along a random vector passing through the current centroid of each cluster and applies K-means on each cluster separately with number of clusters as 2 this time. Now we compare the BICs of the new 2-centroid clusters produced with their respective 1-centroid cluster. If the BIC of the new 2-centroid clusters is higher than the 1-centroid cluster, then the 1-centroid cluster is replaced with these 2 new clusters. The whole process can be viewed graphically in figures 1 to 4.

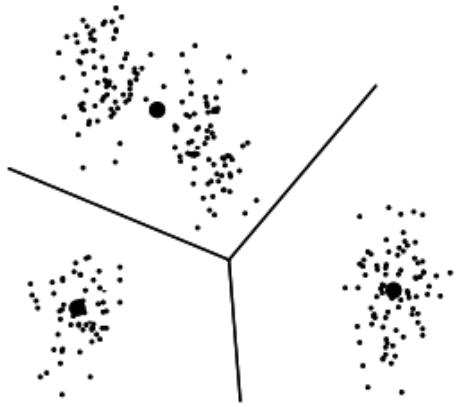


Fig 1. Dataset divided into three clusters using K-means

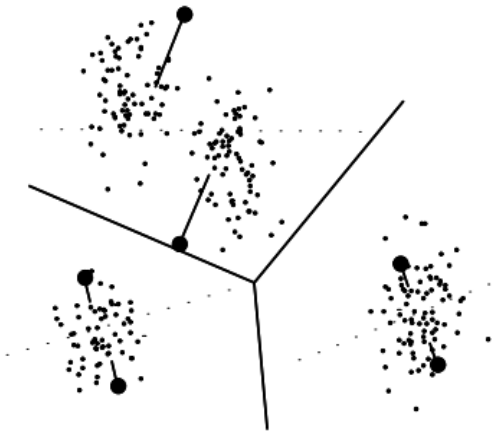


Fig 2. Each cluster is now assigned two centroid and K-means is again applied on all three clusters resulting in centroids settling at their respective positions.

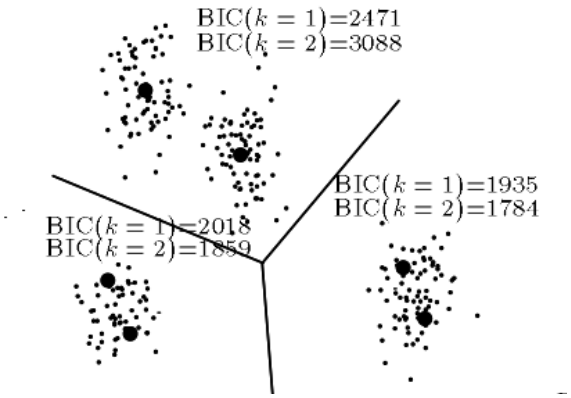


Fig 3. Using the new centroids, each existing cluster is broken into two and BIC values are computed for the newly created clusters.

Fig 4. Final clusters when the Improve-Structure algorithm has completed.

The code executing x-means is present at code_snippet1. Lines 9-24 explain the calculation of BIC, line 33-45 of Improve-Params, and lines 45-71 Improve-Structure. Another function in the code is for plotting the clusters hence obtained, the basic output of the function, in an HTML file, which will be used while illustrating the CAD algorithm in the next section.

The following is the image representation of the result obtained after the first clustering. The interactive HTML cluster can found on this link.

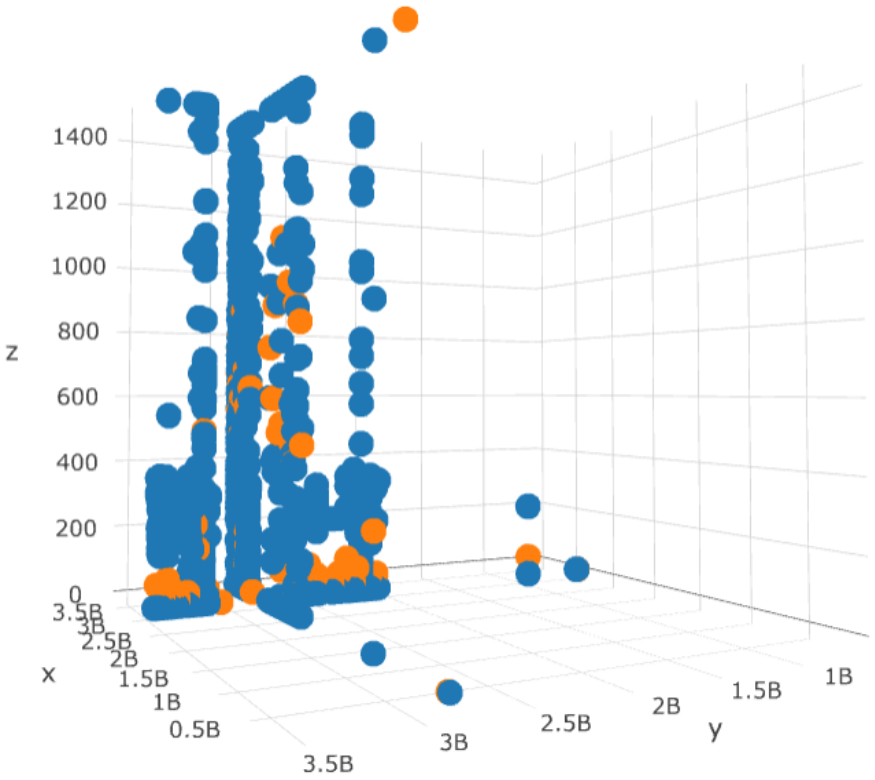


Fig 5. Clustering using X-means
(The blue dots represent the data points (~100,000) and the yellow dots represent calculated centroids)
(x-axis: Source IP, y-axis: destination IP, z-axis: Payload Length)

4. Collective Anomaly Detection

Algorithm

Collective Anomaly detection works on the principle that during DoS attacks, the payload length is maintained at one of the extremes, with it taking up the maximum bandwidth in case of server request attacks and zero payload while flooding the data bandwidth with SYN messages in Neptune attacks. The algorithm uses X-means clustering twice, first on all four attributes and then on the basis of payload length which will be the deciding factor in coming up with the collective anomaly. After the clustering on the basis of the payload length the variance is calculated of the sub-clusters. This way after getting targeted source and destination IPs from the first clustering, we separate out clusters having payloads in a close range to each other in the second clustering. From the sub-clusters, the minimum variance cluster is called an anomaly because as stated earlier, DoS attacks keep the payload length stable, at extremes, so as to keep the bandwidth usage maximum and because varied payloads require another level of difficulty.

1. X-means clustering on the dataset based on <SourceIP, DestinationIP, Payload, Protocol>
2. Sort clusters based on size
3. For clusters i to j
4. Perform x-means on the payload length of data points within the cluster
5. Calculate variance of all the sub-clusters
6. Anomaly[i] = cluster with min variance
7. CAD[i] = <SourceIP, DestinationIP, Payload, Protocol> of Anomaly[i]
8. End
9. Return $\bigcup_{i=1}^{n}$ (CAD[i])

As an example, in the following dataset, we have one cluster from after the first clustering based on the <SourceIP, DestinationIP, Payload, Protocol>. Within this protocol there are two different ranges of payload length, as can be seen. Within the CAD algorithm, these will be clustered somewhat like, c1 = (1460, 1460, 1460, 1460, 1460, 1460) and c2 = (54, 100, 98, 30, 51). Clearly, c1 has 0 variance and hence is the least variance cluster amongst the two. This cluster will be defined as anomaly and the tuple <129.13.46.72, 132.54.67.92, 1460, TCP> will be returned as the collective anomaly detected.

Source IP	Destination IP	Payload Length	Protocol
129.13.46.72	132.54.67.92	1460	TCP
129.13.46.72	132.54.67.92	1460	TCP

129.13.46.72	132.54.67.92	1460	TCP
129.13.46.72	132.54.67.92	1460	TCP
129.13.46.72	132.54.67.92	1460	TCP
129.13.46.72	132.54.67.92	1460	TCP
129.13.46.187	132.54.67.92	54	TCP
129.13.46.187	132.54.67.92	100	TCP
129.13.46.187	132.54.67.92	98	TCP
129.13.46.187	132.54.67.92	30	TCP
129.13.46.187	132.54.67.92	51	TCP

Implementation

The project has been implemented in its entirety and is available for anyone who wishes to study it, work on it, or evaluate it. The github repository contains all the files required for the same. The main function is contained in the file cad.py. The x-means function which is called by the CAD function is written under the file xmeans.py. There are supporting files for data conversions as mentioned in section 3.

5. Result

When the program is run over a dataset of 100,000 data points, we obtained a result which is a collection of 59 different sets, representing the initial clusters based on the 4-tuple, in JSON format. Each set contains a list of 4-tuples representing the actual collective anomaly detected inside the loop which ran over every set of clusters.
We verified our result with the actual attacks as published by MIT Lincoln Lab. We were able to achieve ~95% in detecting DOS attacks.

A part of the output of our algorithm, for the first two and the last three clusters, looks like the following:

Potential DOS attack Detected with records as follows

[[2853066020 3437282560 1460 6]

[2853066020 3437282560 1460 6]

[2853066020 3437282560 1460 6]

[2853066020 3437282560 1460 6]

[2853066020 3437282560 1460 6]

[2853066020 3437282560 1460 6]

[2853066020 3437282560 1460 6]

[2853066020 3437282560 1460 6]]

Potential DOS attack Detected with records as follows

[[167121940 2853065510 0 6]

[167121940 2853065510 0 6]

[167121940 2853065510 0 6]

...

[149819904 2853065510 0 6]

[149819904 2853065510 0 6]

[149819904 2853065510 0 6]]

Potential DOS attack Detected with records as follows

[[2853066118 3238153505 1460 6]

[2853066118 3238153505 1460 6]

[2853066118 3238153505 1460 6]

[2853066020 3238153505 1460 6]

[2853066020 3240901698 1460 6]

[2853066020 3240901698 1460 6]

[2853066020 3240901698 1460 6]]

Potential DOS attack Detected with records as follows

[[3112924672 2853066230 0 6]

[3112924672 2853065609 0 6]]

Potential DOS attack Detected with records as follows

[[2853066177 3367012352 0 6]

[2853066177 3367012352 0 6]

[2853066177 3367012352 0 6]

[2853066312 3367012352 320 6]

[2853066312 3367012352 0 6]

[2853066838 3367411937 0 6]

[2853066312 3354053762 0 6]

[2853066312 3354053762 0 6]

[2853066312 3354053762 0 6]

[2853066312 3354053762 0 6]

[2853066312 3354053762 0 6]

[2853066681 3354053762 0 6]

[2853066681 3354053762 0 6]

[2853066681 3354053762 0 6]

[2853066681 3354053762 0 6]]

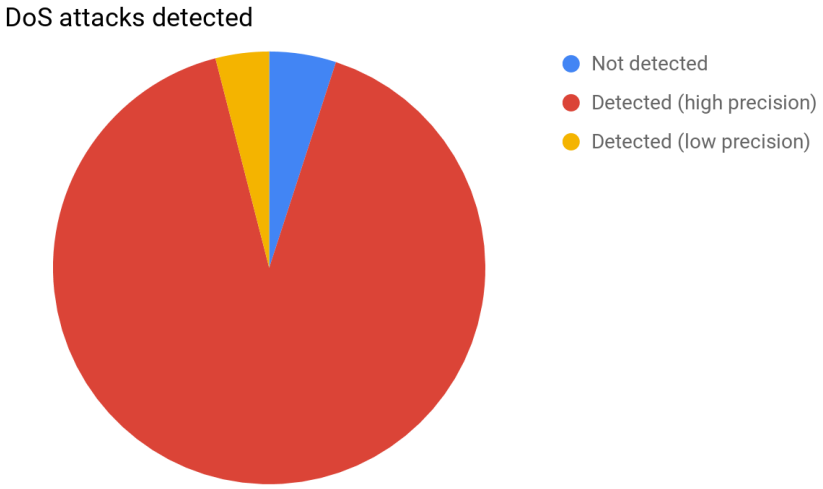
The complete output can be seen in the github repository:
https://github.com/Mrigesh08/Intrusion-Detection-System/blob/master/out2

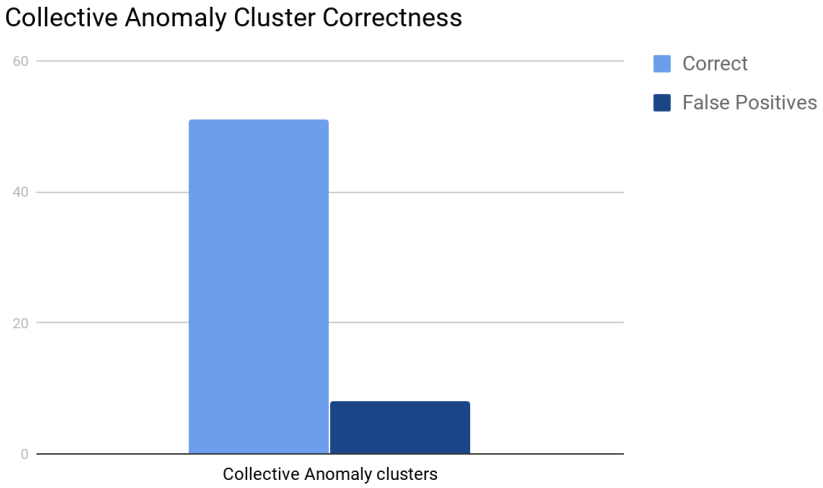
The first column represents the source IP, second represents the destination IP, third the payload length, and fourth the protocol.

The first cluster shows a “back” DOS attack while the second cluster shows a “neptune” DOS attack. Due to the cluster being too large, just a part of the second cluster is shown. As we can

see there is a record [2853066312 3367012352 320 6] in the last cluster which does not correspond to any DoS attack as there is not much resemblance between that and the other data points. Instances like this are false alarms and arise because the limiting K values in the x-means algorithm. The “anomaly” detected corresponding to the second cluster in this data can also be wrong as there is only one instance of the exact tuple. The two tuples have different destination IPs. This is again a false alarm, generated because of co-incidence of having more than one data points of the same payload length in a less dense cluster.

After pruning the result for false positives, against the results of the MIT project, we come up with the result that 8 clusters out of the 59 were false positives, i.e., were in totality not anomalies, or contained more than 20% false positives. These 8 clusters were the ones with no particular attacks. Out of all the back and neptune attacks in the data-set, approximately 91% of all the attacks were detected, with a maximum of 5% false positives and a total of 95% with a maximum of 20% false positives. As for the attacks which could not be detected, the cause was the presence of two or more collective anomalies in a cluster and hence only one was returned.





6. Suggested Improvements

1. Problem with sorting clusters based on size:

The authors have proposed to sort the clusters formed after (applying K-means algorithm) based on size before applying X-means on every cluster thereby giving preference to clusters which have lesser number of datapoints in them. We find this reasoning to be anomalous. Sorting the clusters based on size adds an overhead of O(K*log(K)) computations. In line number 3 of the Collective Anomaly Detection Algorithm proposed by authors, we can see that they are applying the X-means algorithm on all the clusters produced. Hence, sorted or not-sorted, every cluster will be re-clustered based on payload length. This proves that sorting based on size didn't fetch good results. Another important reason to "not sort" the clusters based on size is that DOS attacks require humongous number of requests to be sent within a short period. Due to this, after preliminary clustering, a cluster which represents a DOS attack will have a very large number of datapoints which, after sorting, will fall in the middle of total clusters. The authors, however, gave preference to clusters with small number of datapoints to be anomalous, which seems absurd. The code that we have written omits sorting and the results that we have obtained are exactly same to what the authors have obtained.

2. Dealing with Distributed DoS attacks:

In a distributed attack the basic principle of the DoS attacks of a similar ranged payload length, destinationIP and protocol remains the same, while the attacker's SourceIP becomes variable. Hence, the method suggested by us is to give priority to the first two over SourceIP. We introduce the concept of weights for the same with Payload and DestinationIP getting higher weights than the other two. This way, the initial cluster will have multiple source IP data points which have a similar payload length and the destination IPs. After the initial clustering, we follow the same CAD algorithm as before on a more skewed (towards same target systems) initial clusters.

Modified algorithm incorporating above changes:

1. Assign high weights to DestinationIP and Payload and low to SourceIP and protocol
2. X-means clustering on the dataset on <SourceIP, DestinationIP, Payload, Protocol> with the assigned weights
3. For clusters i to j
4. Perform x-means on the payload length of data points within the cluster
5. Calculate variance of all the sub-clusters
6. Anomaly[i] = cluster with min variance
7. CAD[i] = <SourceIP, DestinationIP, Payload, Protocol> of Anomaly[i]
8. End
9. Return $\bigcup_{i=1}^{n}$ (CAD[i])

7. Code Snippets

1. X-means

```
from sklearn.cluster import KMeans
from scipy.spatial import distance
import numpy as np
import plotly as py
import plotly.graph_objs as go

d=4 # number of dimensions/features to be clustered

def calculateBIC(datapoints, k, centers):
	# datapoints is a 3-d array of datapoints of 'k' clusters
	# 'centers' a 2d array of cluster centers
	global d
	N=0
	for i in range(len(datapoints)):
		N=N+datapoints[i].shape[0]

	# don't split clusters of size less than 3
	if N<3:
		return 999999999

	x=[datapoints[i].shape[0] for i in range(k)] # number of points in each cluster
	const_term = 0.5 * k * np.log(N) * (d+1)
	var = (1.0/((N-k)*d)) * np.sum([np.sum(distance.cdist(datapoints[i],np.array([centers[i]]),'euclidean') ** 2) for i in
range(k)])
	BIC = np.sum([x[i]*np.log(x[i]) - x[i]*np.log(N) - (N*d)*0.5*np.log(2*np.pi*var) - d*0.5*(x[i]-1) for i  in range(k)]) -
const_term
	return BIC

def xmeans(datapoints):
	bestBIC = -1
	bestBICNumberOfCluster =1
	KMAX=int((datapoints.shape[0]/2)**0.5)
	k=2 # starting number of clusters
	while k<KMAX:
		# Improve params
		kmeans = KMeans(n_clusters =k,init="k-means++",max_iter=300).fit(datapoints)
		centers = [kmeans.cluster_centers_]
		labels = kmeans.labels_
		m = kmeans.n_clusters
		n = np.bincount(labels)

		dArray = np.array([datapoints[np.where(labels == i)] for i in range(m)])
```

```
                newBIC = abs(calculateBIC(dArray,m,centers[0]))
                if newBIC > bestBIC:
                        bestBIC = newBIC
                        bestBICNumberOfCluster = k

                # Improve Structure
                # split each cluster into two and calculate BIC
                flag =0
                for i in range(m):
                        if k < KMAX:
                                newdataset=datapoints[np.where(labels==i)]
                                if newdataset.shape[0]>2:
                                        # print("calculating BIC for %d number of records",newdataset.shape[0])
                                        BICone=abs(calculateBIC([newdataset], 1, centers[0]))

                                        kmeans2=KMeans(n_clusters =
2,init="k-means++",max_iter=1000).fit(newdataset)
                                        newLabels=kmeans2.labels_
                                        newCenters = [kmeans2.cluster_centers_]

BICtwo=abs(calculateBIC([newdataset[np.where(newLabels==0)],newdataset[np.where(newLabels==1)]], 2,newCenters[0] ))

                                        if np.isnan(BICtwo):
                                                return bestBICNumberOfCluster
                                        # print("BICtwo=",BICtwo, " BICone=",BICone)
                                        if BICtwo > BICone:
                                                flag=1
                                                k=k+1
                        if flag==0:
                                k=k+1

        # print("FINAL NUMBER OF CLUSTERS=",bestBICNumberOfCluster)
        return bestBICNumberOfCluster

def plot(datapoints, centers):
        trace = go.Scatter3d(
                        x=np.array([datapoints[i][0] for i in range(datapoints.shape[0])]),
                        y=np.array([datapoints[i][1] for i in range(datapoints.shape[0])]),
                        z=np.array([datapoints[i][2] for i in range(datapoints.shape[0])]),
                        mode='markers'
                )
        trace1 = go.Scatter3d(
                        x=np.array([centers[i][0] for i in range(centers.shape[0])]),
                        y=np.array([centers[i][1] for i in range(centers.shape[0])]),
                        z=np.array([centers[i][2] for i in range(centers.shape[0])]),
                        mode='markers'
                )
        data=[trace,trace1]
        fig=go.Figure(data=data)
        py.offline.plot(fig)
```

```
print("PLOT GENERATED")
```

2. CAD

```
import xmeans as xmeans
import pythonarr2 as data
from sklearn.cluster import KMeans
import numpy as np

datap=np.array(data.arr)

def collectiveAnomalyDetection(datap):
        k=xmeans.xmeans(datap)
        kmeans=KMeans(n_clusters=k,init="k-means++",max_iter=300).fit(datap)
        centers = kmeans.cluster_centers_
        labels = kmeans.labels_
        m = kmeans.n_clusters
        n = np.bincount(labels)
        clusterDatapoints=[]
        for i in range(m):
                newDataPoints1=datap[np.where(labels==i)]
                # create an array of the payload length attribute
                newDataPoints2 = np.array([newDataPoints1[x][2] for x in range(len(newDataPoints1))])
                newDataPoints = newDataPoints2.reshape(-1,1)
                k2=xmeans.xmeans(newDataPoints)
                kmeans2 = KMeans(n_clusters=k2,init="k-means++",max_iter=300).fit(newDataPoints)
                # centers2 = kmeans2.cluster_centers_
                labels2 = kmeans2.labels_
```

```
        m2=kmeans2.n_clusters
        n2=np.bincount(labels)
        minVariance = 99999999999
        minVarianceClusterIndex = -1
        for j in range(m2):
                # calculate variance of each of the clusters. and record the label of the cluster with the least
variance
                dataPoints = newDataPoints2[np.where(labels2 == j)]
                newVar = np.var(dataPoints)
                if newVar < minVariance:
                        minVariance = newVar
                        minVarianceClusterIndex = j
        print("Potential DOS attack Detected with records as follows")
        print(newDataPoints1[np.where(labels2 == minVarianceClusterIndex)])


collectiveAnomalyDetection(datap)
```

8. References

[1] Network traffic analysis based on collective anomaly detection - Ahmed and Mehmood, Industrial Electronics and Applications (ICIEA), 2014 IEEE 9th Conference on 9 June 2014
[2] DARPA Intrusion Detection dataset - https://www.ll.mit.edu/ideval/data/
[3] Lincoln Laboratory, Massachusetts Institute of Technology - https://www.ll.mit.edu/
[4] Github, the world's leading software development platform - https://github.com
[5] Denial-of-service attack, Wikipedia - https://en.wikipedia.org/wiki/Denial-of-service_attack
[6] Anomalous Payload-based Network Intrusion Detection - K.Wang and S.J.Stolfo, Springer 2004
[7] Bayesian Information Criterion - https://github.com/bobhancock/goxmeans/blob/master/doc/BIC_notes.pdf
[8] X-means - https://www.cs.cmu.edu/~dpelleg/download/xmeans.pdf