ONLINE SOCIAL NETWORKING

(DETECTION OF MALICIOUS NODES IN A NETWORK)

CONTENTS Page Number

1.OBJECTIVE 3

2. PROBLEM STATEMENT: DETECTION OF MALICIOUS NODES 3

3. BACKGROUND 3

3.1 GRAPH 3

3.1.1 UNDIRECTED GRAPH 3

3.1.2 DIRECTED GRAPH 4

3.1.3 NODE 4

3.1.4 DEGREE 4

3.1.5 EDGES 4

3.1.6 POWER LAW GRAPH 4

3.1.7 HONEST REGION 5

3.1.8 SYBIL REGION 5

3.1.9 SOCIAL NETWORK 5

3.2.0 EGO GRAPH 5

3.2.1 B-MATRIX 5

3.2.2 LOCAL EFFECIENCY 6

4.IMPLEMENTATION 7

4.1. Creation of Honest Region 7

4.2 Creation of Sybil Region 8

4.3 Creation of Social Network 9

4.4 Creation of B-Matrix 11

4.5 Calculating B-Matrix Efficiency 11

4.6 Calculating Local efficiency: 13

5. RESULTS 17

6. REFERENCES 19

1.OBJECTIVE

In the modern era social media has become an important part in the everyday life. Individuals as wells as institutions are relying more heavily on social media for their communication as well as entertainment purposes. If Facebook was a country, it would be the most populous on earth with 1.36 billion active users. Such complex stochastic networks are prone to attacks by unwanted entities who try to obtain certain crucial data to make a profit. Hence it is imperative to detect any attacks on such networks in order to safeguard the user's information. The main objective of this project is to detect the attacks by any malicious user on a network (for instance any social network like Facebook, twitter or any other distributed networks).

2.PROBLEM STATEMENT: DETECTION OF MALICIOUS NODES

Many social network sites like Facebook, twitter or LinkedIn employ a set of proactive monitoring technologies and behavioral analytics to detect any suspicious activity from the accounts in order to find any attacks. But cybercriminals continue to find a way to dupe the system to spread phishing activity and harvest as much user data as possible. This is because the users are attacked using the links that are sent from the people of their circle. The malicious user can easily disguise himself as an honest one mimicking their network structures. This further increases the complexity of detecting the attacks since most detection methods assume that the network structure of an attacker is sparse as compared to user. These detection methods use the various attributes associated with a network to identify any attacker using machine learning techniques. They do not consider the probability that the attacker can also have the same network properties as the user. Hence the need to find the method to detect an attack of such malicious users who use the same network structure as any honest user becomes imperative.

3.BACKGROUND

3.1 GRAPH

Graph is a collection of nodes [1]. The nodes are connected by lines called edges. The graph may be a directed or undirected.

3.1.1 UNDIRECTED GRAPH

Undirected graph is one in which the nodes are connected together and all the edges are bidirectional. The nodes and edges are unordered pairs of elements in this graph.

3.1.2 DIRECTED GRAPH

Directed graph is one in which the edges in the graph are pointing only in one direction.

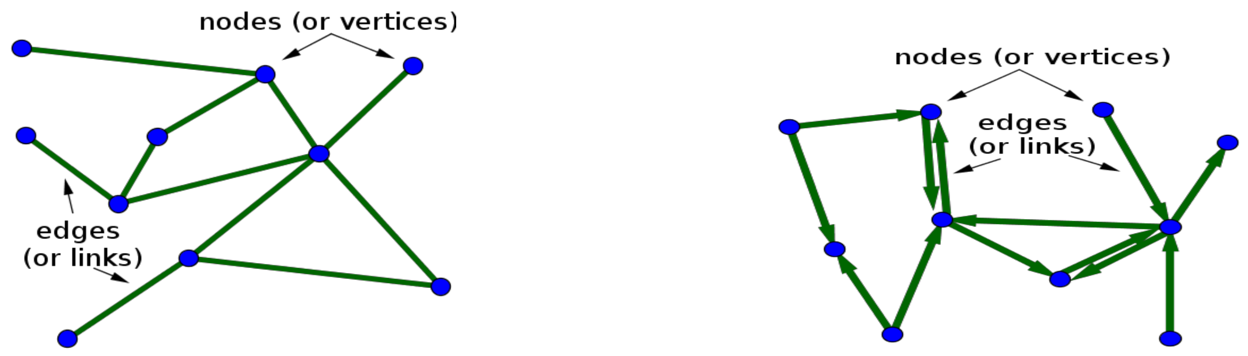

Figure1

Undirected Graph

Figure2

Directed Graph

Figure 1: Directed and Undirected graph

In this project only undirected graph is used to simulate online social network containing the honest and the Sybil regions.

3.1.3 NODE

A node is a vertex of a network. These vertices are linked to form the network.

3.1.4 DEGREE

The node degree is the number of edges that the node has.

3.1.5 EDGES

An edge of a network represents the connection between the nodes in the network. The edges can be directed or undirected.

3.1.6 POWER LAW GRAPH

The benchmark graph in this project is created using Power Law Graph. A power law graph is one in which one quantity varies as a power of another. The powerlaw_graph function in Network X is used to create a graph with power law degree distribution and approximate average clustering. It is essentially the Barabási–Albert (BA) growth model with an extra step that each random edge is followed by a chance of making an edge to one of its neighbors too. This algorithm improves on BA in the sense that it enables a higher average clustering to be attained if desired.

3.1.7 HONEST REGION

The honest region is a network that is used to simulate the network of the honest users (without malicious nodes). It is an undirected graph that is created using the honest_region function from the sypy library. This function can be used to create various graph structures like small world graphs, power law graphs etc. Here the power law graph function is utilized to create the honest graph to create a power law distribution of nodes which closely resembles any real-world networks. All the nodes of this graph have the attribute ‘honest’.

3.1.8 SYBIL REGION

The sybil region is a network that is used to simulate the network of the Sybil users (with malicious nodes). It is an undirected graph that is created using the sybil_region function from the sypy library. Here also the power law graph function is utilized to create the sybil graph to create a power law distribution of nodes like the honest graph. All the nodes of this graph have the attribute ‘sybil’.

3.1.9 SOCIAL NETWORK

The honest and the sybil graphs are stiched together using the function social_network which is available on the sypy library that is used to create the attack edges from the sybil to the honest region. As a result both of them are combined into a single graph. The number of attack edges can be specified in order to simulate a real world attack on the network.

3.2.0 EGO GRAPH

Ego graph of a network is a subgraph of neighbors centered at node n within a given radius. It is used to calculate the local efficiency of a graph.

3.2.1 B-MATRIX

B-matrix is a matrix formed with number of shells(l) in the network as the rows and number of (k) members in the column. Each entry in the matrix corresponds to the number of nodes that have k members in their respective l shells. The B-matrix is then read and efficiency for each node in the matrix is found using the formula:

$\frac{1}{N(N-1)}$ Σ(i,j)$\frac{1}{d(i,j)}$ if(i!=j), where N is the number of nodes in the ego-graph and d(I,j) is the shortest distance between the two nodes i and j.

3.2.2 LOCAL EFFECIENCY

The efficiency of a pair of nodes in a graph is the multiplicative inverse of the shortest path distance between the nodes. The local efficiency of a node in the graph is the average global efficiency of the subgraph induced by the neighbors of the node. The average local efficiency is the average of the local efficiencies of each node

The two networks are stitched together by the attack edges from the Sybil network. As an analogy the honest network can be considered as a Facebook network of the user and the Sybil network can be considered as the Facebook network of the malicious user with the attack edges as any friend or message requests.

In the project the local efficiency of the social network is compared with the B-matrix to find any relation between them. These relations are then use to find the weights that are used to rank the social network to distinguish the Honest and the Sybil regions.

4.IMPLEMENTATION

4.1. Creation of Honest Region

A Power law graph with n nodes is created, each node with node degree 2. The probability of forming a triangle (connecting three nodes) is kept 0.03. To produce the same graph for every iteration the seed is kept 1. For honest region the above graph is implemented with 100 nodes. Here 10 random nodes are selected that are called as known nodes, used to initiate the analysis.

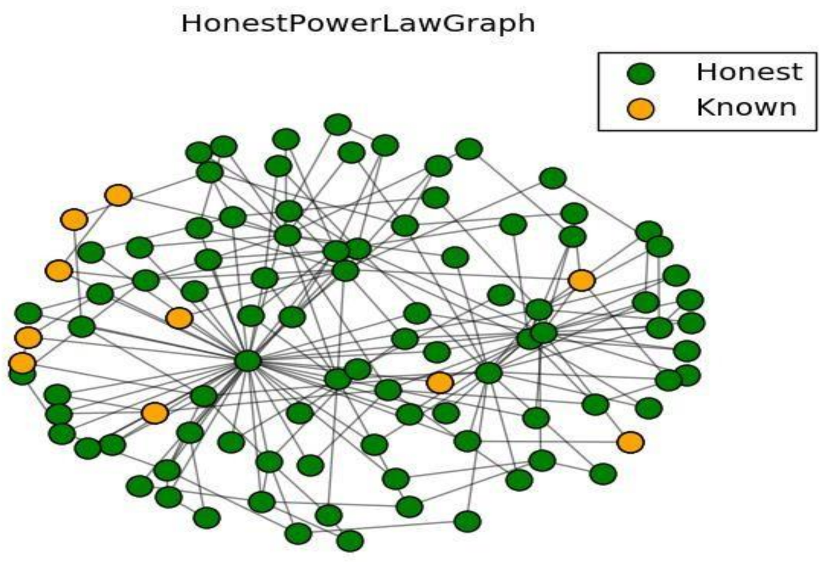

Figure 3

Graph showing the honest network

Python snippet for creating honest region

```
#create honest region
honest_region = sypy.Region(
    graph=sypy.PowerLawGraph(num_nodes=100,node_degree=2,prob_triad=0.03,seed=1),
name="HonestPowerLawGraph")
```

```
honest_region.pick_random_honest_nodes(num_nodes=10)

nx.set_node_attributes(honest_region.graph.structure, 'Node_type', 'Honest_node')
```

Here 10 random nodes are selected which are used to initiate the analysis.

4.2 Creation of Sybil Region

Sybil region the graph is implemented with 80 nodes using the same power law graph. For producing the same network for every iteration, the attribute seed of the sybil_region is kept at 1.The graphs are created with the above specifications and are then plot using visualize function from the SYPY library.

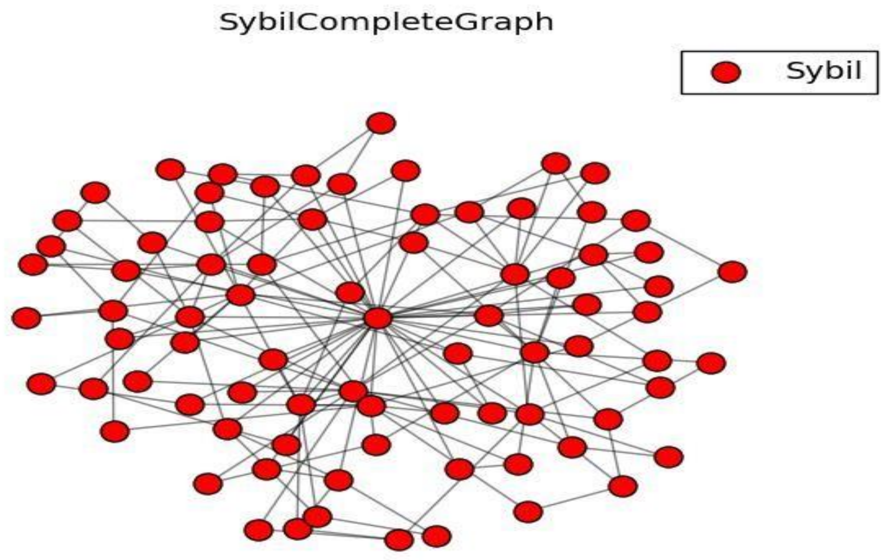

Figure 3

Graph showing the sybil network

Python snippet for creating sybil region

```
#create sybil region
```

```
sybil_region = sypy.Region(
    graph = sypy.PowerLawGraph(num_nodes=80,node_degree=2,prob_triad=0.03,seed=1),
    name = "SybilCompleteGraph",is_sybil=True)
nx.set_node_attributes(sybil_region.graph.structure, 'Node_type', 'Sybil_node')
```

4.3 Creation of Social Network

From the above two graphs 10 random nodes are picked and paired between the Sybil region and the honest region. This striching is used for combining the Sybil and honest region into one social network region. The created graph is then converted in to edgelists and stored as a .txt file.

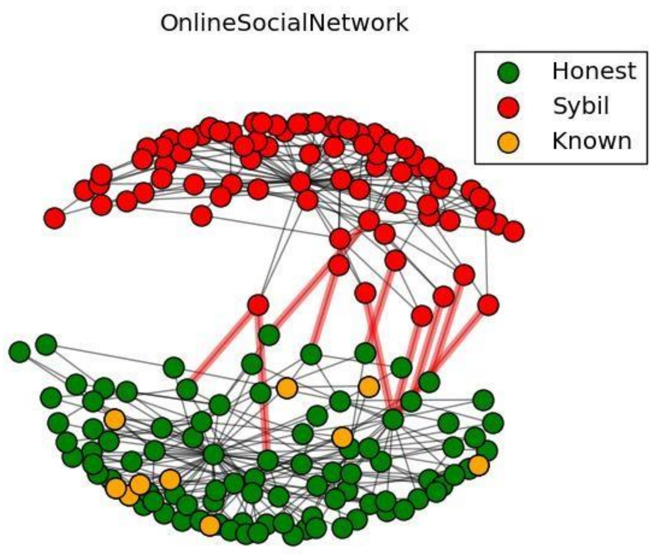

Figure 4

Graph showing the social network

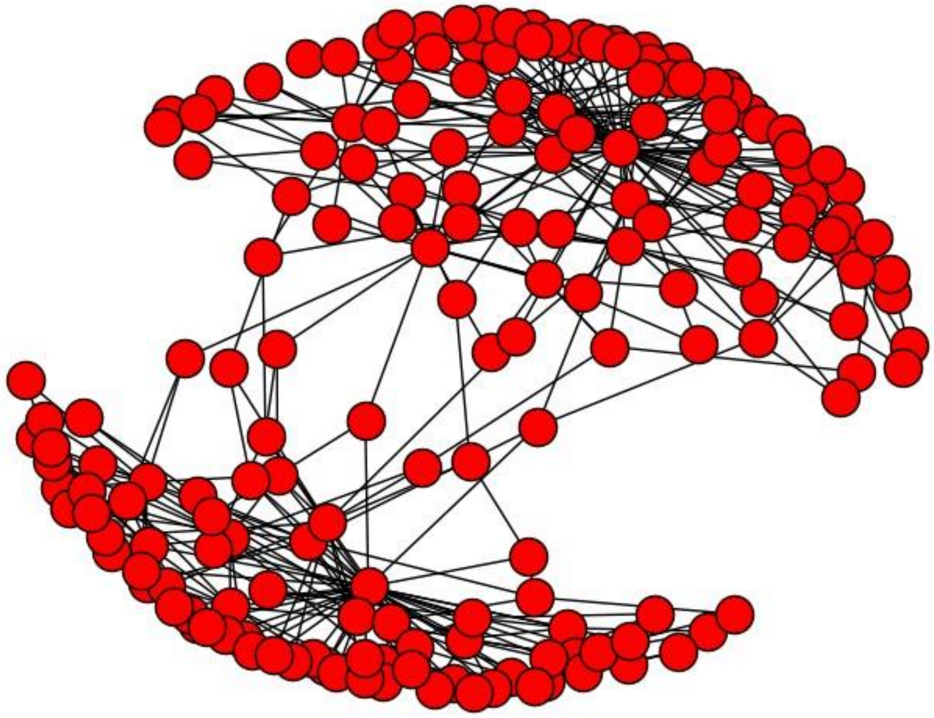

Figure 5

Graph showing the social network imported to networkx

Python snippet for creating sybil region

```
#create online social network

social_network =
sypy.Network(left_region=honest_region,right_region=sybil_region,name="OnlineSocialNetwork")

social_network.random_pair_stitch(num_edges=10)

social_network.visualize()

G = social_network.graph.structure

#print G.nodes(data='True')
```

nx.draw(G)

plt.show()

4.4 Creation of B-Matrix

B-matrix is a matrix formed with number of shells(l) in the network as the rows and number of (k) members in the column. The above edge list is taken as the input to create the B-Matrix. Each entry in the matrix corresponds to the number of nodes that have k members in their respective l shells.

4.5 Calculating B-Matrix Efficiency

The B-matrix is then read and efficiency for each node in the matrix is found using the formula:

$\frac{1}{N(N-1)}$ Σ(i,j) $\frac{K}{l}$B(i,j) if(i!=j), where N is the number of nodes in the graph and B(I,j) is the entry in B-matrix. K is the members in the rows of B-matrix and L is the shells in the B-matrix.

Python snippet for calculating B-matrix Efficiency:

```
for k in range(1,180):
    for l in range(1,8):
        s=s+((k/l)*arrmat[l][k])
```

The B-matrix efficiency is obtained in the terminal window.

Bmatrix_efficiency:

```
[0.38569164649219867,     0.39542490266546876,     0.31929186579729341,
0.38569164649219867,      0.67037248592626453,     0.69281057326511886,
0.72864041169125926,      0.31572435891687672,     0.68145302288372345,
0.78090432571103285,      0.69281057326511886,     0.59985745520629252,
0.4623832287097594,       0.39542490266546876,     0.69281057326511886,
0.72864041169125926,      0.71639435435191046,     0.40043028118022156,
0.72864041169125926,      0.71639435435191046,     0.83899996028436408,
0.72864041169125926,      0.78090432571103285,     0.71639435435191046,
0.71639435435191046,      0.72864041169125926,     0.72864041169125926,
```

0.72864041169125926, 0.42694208996730015, 0.79484904581301563,
0.69281057326511886, 0.72864041169125926, 0.70445444844604521,
0.61882440476190492, 0.78090432571103285, 0.72864041169125926,
0.42694208996730015, 0.72864041169125926, 0.4623832287097594,
0.83899996028436408, 0.72864041169125926, 0.87050965072771946,
0.4623832287097594, 0.93909722963121045, 0.88693436111880852,
0.43828901724509961, 0.88693436111880852, 0.72864041169125926,
0.43828901724509961, 0.72864041169125926, 0.37175201492274673,
0.41073042315080538, 0.71639435435191046, 0.87050965072771946,
0.92120966335252064, 0.79484904581301563, 0.72864041169125926,
0.97647151269748844, 0.72864041169125926, 0.32292018245408083,
0.87050965072771946, 0.72864041169125926, 0.80917065024207901,
0.32292018245408083, 0.72864041169125926, 0.71639435435191046,
0.4623832287097594, 0.45617674241835326, 0.72864041169125926,
0.71639435435191046, 0.99600094295143815, 0.70445444844604521,
0.41073042315080538, 0.92120966335252064, 0.71639435435191046,
0.80917065024207901, 0.83899996028436408, 0.82388284388284405,
1.1507217483195522, 0.40553130387041547, 0.41073042315080538,
0.42694208996730015, 0.4623832287097594, 0.4623832287097594, 0.88693436111880852,
0.88693436111880852, 0.33418409154905077, 0.71639435435191046,
0.43255974904581723, 0.71639435435191046, 0.79484904581301563,
0.69281057326511886, 0.79484904581301563, 0.4623832287097594,
0.71639435435191046, 0.71639435435191046, 0.72864041169125926,
0.88693436111880852, 0.82388284388284405, 0.70445444844604521,
0.36727307498391842, 0.75409366745198847, 0.82388284388284405,
0.44413287080836761, 0.38569164649219867, 0.82388284388284405,
0.32661069882498461, 0.97647151269748844, 0.61882440476190492,
1.0368593613491575, 0.82388284388284405, 0.40553130387041547,
0.31572435891687672, 0.87050965072771946, 1.2844240965027485,
0.87050965072771946, 0.88693436111880852, 1.0582379048821298,
0.87050965072771946, 0.31572435891687672, 0.93909722963121045,
0.87050965072771946, 1.0368593613491575, 0.3460477992366538, 0.44413287080836761,
0.87050965072771946, 0.87050965072771946, 0.88693436111880852,
0.48178206531654816, 1.4089088968920904, 0.92120966335252064,

0.87050965072771946, 0.87050965072771946, 0.67037248592626453, 1.175737438500412, 0.87050965072771946, 1.4089088968920904, 0.71639435435191046, 1.4428585088653938, 0.33806995307869087, 0.88693436111880852, 0.53991034369952373, 0.68145302288372345, 1.6325234540424418, 0.32661069882498461, 0.68145302288372345, 1.1030273600668339, 0.88693436111880852, 0.88693436111880852, 0.88693436111880852, 1.0368593613491575, 0.87050965072771946, 0.54790901545803516, 0.53991034369952373, 1.4089088968920904, 1.0368593613491575, 0.88693436111880852, 1.2844240965027485, 0.88693436111880852, 1.2844240965027485, 1.0802845279005076, 1.2844240965027485, 0.42694208996730015, 0.3460477992366538, 0.88693436111880852, 0.88693436111880852, 1.4089088968920904, 1.4428585088653938, 0.88693436111880852, 0.88693436111880852, 1.4780501798133303, 1.6325234540424418, 1.4089088968920904, 1.4428585088653938, 0.44413287080836761, 0.32661069882498461, 1.1030273600668339, 1.4428585088653938, 1.6749266606409468, 1.3445080892554833]

4.6 Calculating Local efficiency:

Similarly, the local efficiency for each node in the social network is computed using the metric:

$\frac{1}{[illegible](-1)}$ Σ(i,j) $\frac{1}{[illegible]}$ [illegible] if(i!=j), where N is the number of nodes and d(i,j) is shortest distance between two nodes i and j. H(i,j) is the ego-graph of each node in the graph. A ego-graph is found using the function ego_graph() in networkx. The snippet is given below:

```
H=(nx.ego_graph(G,i,radius=rad,center=False,undirected=True,distance= None)
```

A ego-graph created for the node-1 is shown below:

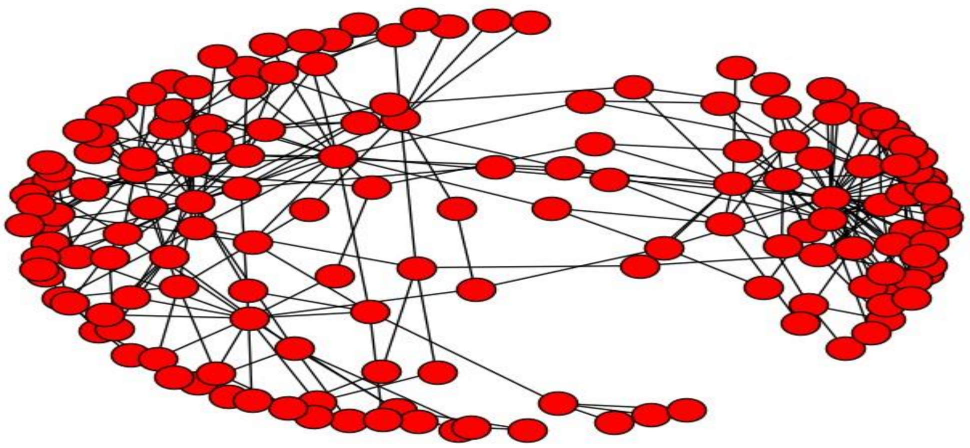

Figure 7

Python snippet for calculating the local efficiency:

```
def efficiency (G, u, v):

    return 1 / nx.shortest_path_length(G, u, v)

def global_efficiency(G):

    n = len(G)

    denom = n * (n - 1)

    return sum(efficiency(G, u, v) for u, v in permutations(G, 2)) / denom

def local_efficiency(G):

    return sum(global_efficiency(G) for v in G) / len(G)
```

The local efficiency values of all nodes in the ego-graph is obtained from the terminal:

local efficiency:

[0.29033223058409224, 0.32066178122187455, 0.30625035896412106, 0.322649081103146, 0.3577702252432223, 0.36183195592285483, 0.36757448452363417, 0.30487130512238436, 0.3537348146140883, 0.37355741862030084, 0.362823691460O455, 0.3429934406678425, 0.3321581785867149, 0.32416984875706795, 0.36390266299356194, 0.3670312424549686, 0.36504534016996115, 0.3196276456605603, 0.366642522574722, 0.36589279779707995, 0.37962746733390146, 0.36739340383407887, 0.3733245355276133, 0.3652684802734595, 0.3654631344062997, 0.36774349316721816, 0.36711333236756627, 0.36740064706166015, 0.33057263309359713, 0.3759797724399547, 0.36098320871047274, 0.36760345743396283, 0.36433706816058936, 0.348501945069352, 0.3727086891269502, 0.36768313293736615, 0.33156292820154004, 0.36769520498333674, 0.33473524544949207, 0.38048095635252804, 0.36776763725915873, 0.3850063459097889, 0.3357479928908109, 0.39146751306948163, 0.3875124904484985, 0.333140591545046, 0.38576970551932727, 0.3677917813511009, 0.33105030788888656, 0.36776763725915884, 0.3223730334705671, 0.3289746700574253, 0.36568864834068654, 0.3845679012345833, 0.3908608058608279, 0.37578750526759813, 0.36776763725915884, 0.3953212968355968, 0.36776763725915884, 0.31220345097463614, 0.3839823468328305, 0.368409870104782, 0.37650418275418945, 0.31261968963240094, 0.36768313293736615, 0.36596638655461733, 0.3345437274008299, 0.3339878469072702, 0.36761070066154444, 0.3652708541043477, 0.39505610561059024, 0.3510387488328697, 0.32888231725806516, 0.39084249084251355, 0.36579072306888355, 0.3776195838695909, 0.3807756463719876, 0.37899262899263914, 0.4055250514756612, 0.32744996410233745, 0.3289746700574264, 0.3314002368623785, 0.334689281117813, 0.33476588833727833, 0.385784400164598, 0.38751249044849845, 0.3125900509695475, 0.36596638655461733, 0.33249811620550174, 0.36596638655461733, 0.3759745048461914, 0.3630073461891549, 0.3752739148756896, 0.33448244162526025, 0.36589279779707984, 0.3667378815933093, 0.3677314211212484, 0.3859284076882566, 0.37878787878788905, 0.36468720821661293, 0.2905930793882299, 0.33151531377168686, 0.3454767754767837, 0.3251489540628136, 0.32034061074773984, 0.34762411762412887, 0.31230504019789135, 0.3584925719462853, 0.3271003233970553, 0.3623563133767362, 0.3488892788892889, 0.31979436618631724, 0.3061057955289281, 0.3513124495211884, 0.3855549880830909, 0.3529479635398823, 0.35357961558809586, 0.3653517076934018, 0.35252682589132855, 0.3068191396827307, 0.3584982574060405, 0.35346717433946634, 0.3620677130881352, 0.3150547938473386, 0.3268243456322412, 0.3536258220837844, 0.35302007615093417, 0.3542085464057075, 0.3295005473453277, 0.394691876750712, 0.3571489621489785, 0.3534498673128135, 0.3528441213799643, 0.3360624004366381, 0.373839577906913, 0.3536113995615724, 0.39517740429506204, 0.3387775449162666, 0.39743736852171385, 0.3148901542065817, 0.35463175218951926, 0.33420203847958363, 0.3355439642324784, 0.40613978145623963, 0.3139191403834245, 0.33602944497130965, 0.3714217836257449, 0.35464644683479146, 0.3533621348380815, 0.35463175218951926, 0.3678416821274126, 0.3534383292950451, 0.33519063180823855, 0.3337673025414153, 0.3956909430438954, 0.3681440252868982, 0.35463175218951926, 0.3862359550561933, 0.3546170575442484, 0.38526557711952264, 0.3679195303551127, 0.3853506979911603, 0.3231516849163404, 0.3164356970858313, 0.35452888967262136, 0.3544260271557221, 0.3950606909430546, 0.3968493019697911, 0.3546170575442484, 0.35381766884149074, 0.40004897639338555, 0.40421399978362177, 0.39522408963586475, 0.39880952380953194, 0.3270612845579214, 0.31302025355596397, 0.37043128654972174, 0.3909734174794482, 0.4084027084027084, 0.38546607095124585]

5. RESULTS

After computing the efficiency for the B-matrix and the local efficiency of each node in the ego graph, the results are plotted against the node degrees. It is observed that the B-matrix efficiency and Local efficiency are consistent and are linearly dependent on each other. From this it can be inferred that by converting a complex network in to B-matrix and then by ranking them with a weighing function, the malicious nodes and the honest nodes can be differentiated. The resultant plots are shown below:

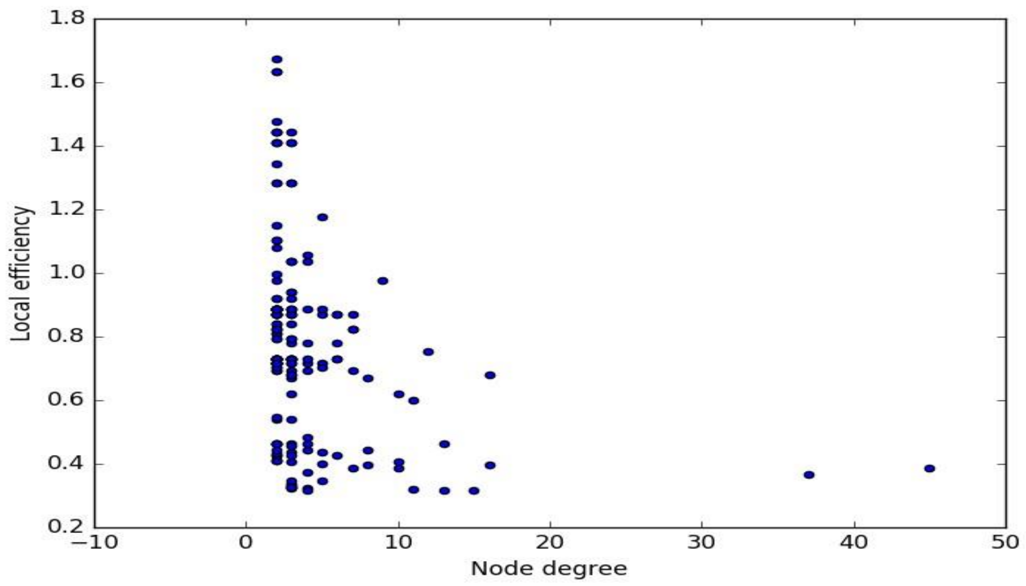

Figure 8

Plot between Node degree and local efficiency

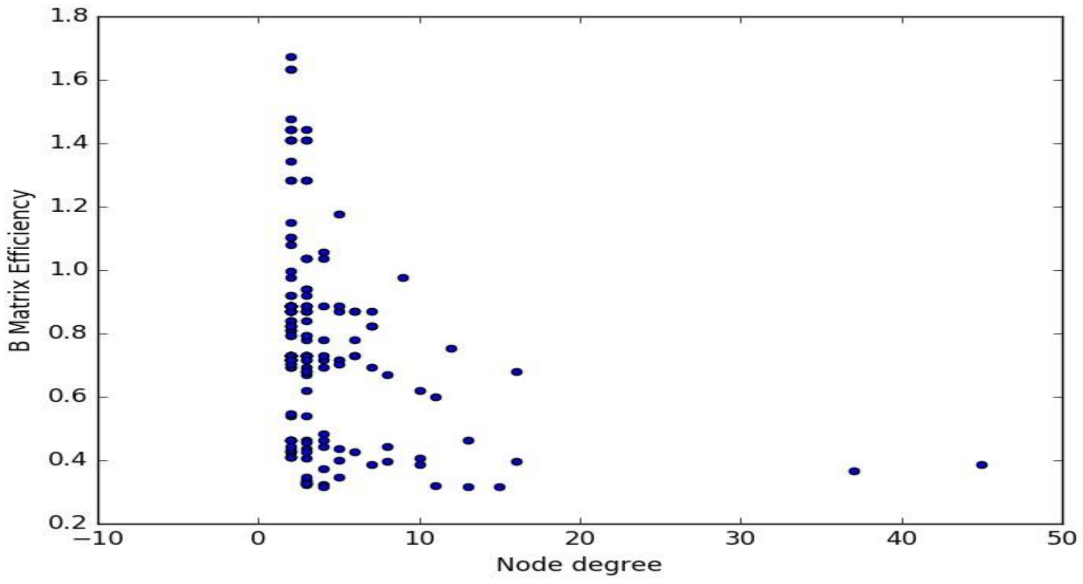

Figure 9

A plot between Node degree and B-matrix efficiency

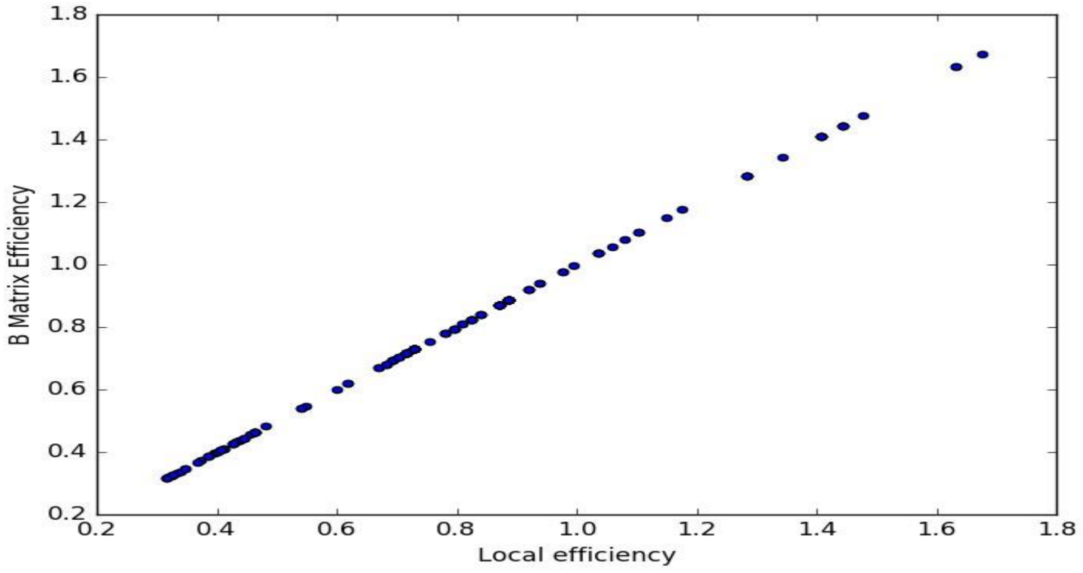

Figure 10

A plot between Local efficiency and B-matrix Efficiency

6.REFERENCES:

[1] https://arxiv.org/pdf/cond-mat/0703470v2.pdf

[2] https://www.usenix.org/system/files/conference/nsdi12/nsdi12-final42_2.pdf

[3] http://www.cs.colostate.edu/~dmulamba/files/summary_of_sybilradar.pdf

[4] https://networkx.github.io/documentation.html

[5] http://matplotlib.org/examples/index.html

[6] http://www.numpy.org/

[7] http://sypy-tony.readthedocs.io/en/latest/overview.html#what-is-sypy

[8] https://docs.python.org/3/tutorial/