Summer Internship Project
Report

Topic

Submitted by

Name of the student
Designation of the student
Institute of the student

Under the guidance of

Name of the guide
Designation of the guide

Department of Physics
NAME OF THE INSTITUTE PROVIDING INTERNSHIP
Address

Summer Internship 20**

Department of Physics

Name of the institute

Certificate

Name of the guide
(Project Guide)

Date:

Abstract

Contents

Chapter 1

Objective

Chapter 2

Introduction

2.1 Graphs and Networks

2.1.1 Introduction

In this first chapter ,we will learn how to create a graph network using networkx module Python .Before proceeding that we will give an introduction to network graphs .We will discuss about the structure of the graph and give a brief introduction to the kinds of models we use in our program.

2.1.2 Graph and Their Degree and Connectivity Structure

In mathematics, networks are often referred to as graphs. A graph $G = (V, E)$ consists of a collection of vertices, called nodes ,where V is a set of N nodes ,and a collection of edges set , E .

The nodes correspond to the objects that we model , so it could be a person ,customer, a country , depending on the use case ,The edges represent the connection between any two objects. When dealing with social networks, the nodes represent the individual, people, or things in the population, while the edges represent the connection or relationship between nodes.

In our Setting , graphs are usually undirected, the way the relationship between nodes is insignificant , thus , an edge is an unordered pair (u,v) ϵ E indicating that u and v are directly connected, when G is undirected, if u is directly connected to v, then also v is directly connected to u. Thus, an edge can be seen as a pair of vertices.

In this book,we only consider finite graphs.,without loops or multiple edges having at least one edge.This means that V is a finite set of size, say , n ϵ N.In this case, by numbering the vertices as 1,2,.......,n, we may as well assume that V=[n]=1,2,3,........n .

We represent a network by a connected graph G . A graph is said to be connected if there is a pathbetween every pair of nodes . From eve node to any other node, there should be some path to traverse. That is called the connectivity of a graph, and may have at most (n–2) cut nodes.

In social Networks context , the degree of a node is the number of its neighbors

. we will often be interested in the distribution of the degree of all the nodes in a network .

The degree du d_u of a node u is equal to the number of edges containing u .

$$d_u = \{v \epsilon V : \{u, v\} \epsilon E\}$$

Plenty of models of networks have been proposed, based on our understanding of real-world interactions .

A model is a simple representation of the real world through some mathematical rules designed to approximate the laws of the real world, and can be used to calculate or predict what might happen .

A simulated network, looks as much as possible as the real-world network we want to model. One of the most important properties of graphs is that they are just abstractions of the real world and that can be used to generate new networks with similar properties. in this report,we will explore two famous models Erdos-Renyi and Small World, and We will learn how to create the two models using the Networx module in Python.

Erdős–Rényi model

Random graphs were introduced by Paul Erdos and Alfred Renyi in 1959 to be able to prove certain results.Random graph is a graph that is generated by a random process.

The Erdős–Rényi model is specified by two parameters: the number of vertices in the graph n, and the probability of an edge p. Given n and p, we choose a graph on n vertices by including an edge between each pair of vertices with probability p, independently for each pair.
The process to generate a Erdos-Renyi model is simple:

A graph $G = (V, E)$ is built, with $V = n$ and such that each pair of nodes has a probability p to have an edge between them. Thus, $E\ [E]$ = p $\cdot \binom{n}{2}$,

and the mean degree is $m = pn$. This model can be extended to directed graphs simply by putting a direction to edges.

The degree of distribution is the probability that a node will have a certain degree d . Since there are $\binom{n}{d}$ ways to choose d nodes from among n total, and p^d probability that they will have edges, $p^d \cdot \binom{n}{2}$is the probability that a node has edges to d nodes. However, there must be no edges to the rest of the $(n-d)$ nodes, which occurs with probability $(1 - p)^(n - d)$

$$P[d] = (n - d)p^d(1 - p)(n - d)$$

In this model, a single giant connected component exists if and only if $p > 1/n$.

if $p < 1/n$, then all connected components of the graph have size O(log(n)) .

Small-world network graphs

Small-world network graphs were introduced by Watts and Strogatz to be able to generate networks similar to real-world networks, This model starts from a regular ring lattice graph, such as the one shown in Figure 2.1 (a), where all the nodes have the same degree and, when placed on a ring, are connected only to their four closest neighbors on the ring.

They start with the lattice model that has high clustering and high mean path length. Then, add to the model a probability p that an edge is rewired, meaning that the edge is disconnected from one of its nodes and then randomly connected to another node anywhere in the network. Each edge is chosen to be rewired independent with probability p.

When the probability p is low, then most connections are still the original local connections that they connect nodes that are nearby in the lattice. For this reason, the transitivity is still high. However, some of the edges that have been rewired might turn into long distance connections that connect nodes that are far away from each other in the lattice. These long distance connections, or shortcuts, immediately create a short distance between the nodes around either end of the shortcut. Implicit in this conclusion is the fact that path length counts each edge as length one, so the shortcut connections add only one to the path length even though we frequently draw them as long connections.

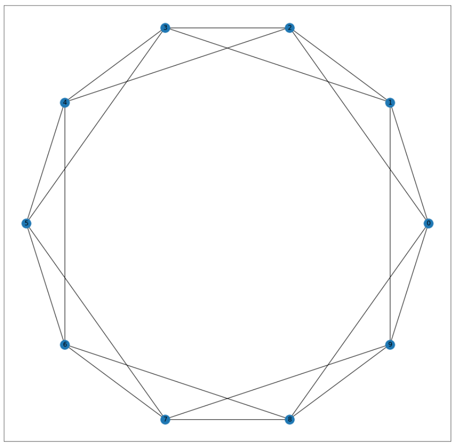
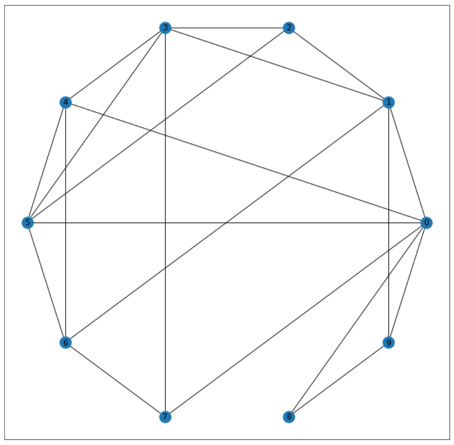
[A]

[B]


Figure 2.1: Small-world network graphs

Small world model network. A network of N=10 nodes spread around a ring. Originally, each node was symmetrically connected to its 4 nearest neighbors along the ring. But then, each edge was rewired with probability p. If an edge was selected for rewiring, one end of the edge was disconnected from a node and reconnected with a randomly chosen node.

Next, we will learn how to implement a graph with python , For implementation of the graph , we will need to install the networkx library as well you'll need to install the matplotlib library. Following you'll see the exact code of the graph which has been used as a function of the networkx library lately in this article.

Python has libraries/packages for dealing with graphs and networks such as NetworkX, igraph, SNAP, and graph-tool. They use similar interfaces for handling and processing Python graph data structures.

We will use the popular NetworkX package of Python.It is used to study large complex networks represented in the form of graphs with nodes and edges. NetworkX includes many generator functions and facilities to read and write graphs in many formats. We can generate many types of random and classic networks, analyze network structure, build network models, design new network algorithms and draw networks. The new version (2.2) of networkx largely covers what is needed to create, manipulate and analyze networks, and It's simple to install .

Installation of the package:

The first thing to do is to install a module named network X :
Using Pip its as easy as:

- *pip install networkx*

Once installed import the package and Initialize a graph object

- *import networkx as nx*

The "networkx" module is now available under the "nx" alias. You can use any alias, although "nx" is the most common alias for the "networkx" module in Python.

To create an empty graph, we use the following command:

- $G = nx.Graph()$

We can add nodes to the network by concatenating the return value of Graph() with *.add-node()* *.add-nodes-from()* , for multiple nodes in a list.

- *G.add-node("A")/G.add-nodes-from()("A", "B", "C", "D")*

We can add an edge connecting two nodes A and B as follows:
We can use*.add-edge()* with the names of two nodes as parameters (or *.add-edges-from()* for multiple edges in a list)

- *G.add-edge(*("A","B"))*

we can access all nodes when using *G.nodes)*
we can access all edges when using*G.edges*

In Python, the networkx package has a built-in function to generate Erdos-Rényi graphs.

Drawing graph

We can easily draw a graph using the 'networkx' module. We use the 'matplotlib' library to draw it. So, we need to import it first.

- *import matplotlib.pyplot as plt*

  Now, the graph (G) created above can be drawn using the following command:

- *nx.draw(G)*
- *plt.show()*

In Python, the networkx package has a built-in function to generate Erdos-Rényi graphs.

The following Python code will create and draw a simple $Erdos_Rényi$ graphs with 50 nodes and p=0.2 the Probability for edge creation.

```
# Generate the graph
n = 50
p = 0.2
G_erdos = nx.erdos_renyi_graph(n,p, seed =100)

# Plot the graph
plt.figure(figsize=(12,8))
nx.draw(G_erdos, node_size=10)
```

++++++++++++++image +++++++++++++++++++++++++++++++++++++
graph Edros-renyi

To generate smart-networks graphs , with $nx.watts_strogatz_graph$ function from the NetworkX package

The following Python code will create and draw a simple $Erdos_Rényi$ graphs with 50 nodes , p=0.2, The probability of rewiring each edge. m=4, The number of neighbors connected to each nodes to create a ring topology.

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.watts_strogatz_graph(n = 10, m = 4, p = 0.5)
pos = nx.circular_layout()

plt.figure(figsize = (12, 12))
nx.draw_networkx(G, pos)
```

++++++++++++++image +++++++++++++++++++++++++++++++++++++
graph smart-network

Chapter 3

Attacker-Defender game model

3.1 Introduction

In the first half of this chapter, we will construct an attack_defense game model based on the SIR model in a complex network to understand the evolution and control of epidemic spreading. We will analyze the evolution process of network system security states.

In the second half of this chapter , we will discuss how to visualize the different decision making by both players over time using python. ....

3.2 Evolution of Security States in Attack and Defense :

In network attack and defense , The attacker exploits the vulnerability of network nodes. Then it infiltrates some (maybe only a few) nodes and infects them with a malware pathogen while others are not infected but susceptible to the disease. . On the other hand, The defense system uses a honeypot to detect suspicious activities and install the patch to get rid of the virus in the nodes , the defense can recover infected nodes and could not pass the pathogen to other susceptible individuals.

According to the actual network attack and defense, the evolutionary states in the SIR model are expanded to three states, and the nodes are divided into three groups, according to their security states,denoted as S (Susceptible), I (Infected), and R (Removed).

S : The Susceptible node is in normal working condition, but the node may be attacked due to its inherent vulnerability.

I : The infected node is penetrated or infected by the attack strategy, the attacker can use this node to attack neighboring nodes.

R: The resistant node is a node that is unable to become infected thanks to a defensive mechanism existing on the system , a node may also be resistant to the malware because the user operating system is not targeted by the malware.

+++++++++++++++++++++Figure 2++++++++++++++++++++++++

In our model,(as shown in Figure 1),the evolutionary states are expanded to three states and there are four migration modes for network node states.

$I \rightarrow S$: in our model SIR , they have two ways to transform the infected nodes to susceptible nodes .

1-Faced with an attack strategy, if the defense strategy succeeds in identifying the infected node and clears the infection. For example, a defender installs a patch to defend against a virus attack. As a result, the defender restricts the attack damage effect that has not appeared yet, avoids the loss of the infected node and transforms the node into the susceptible state.

2- With a recovery rate δ,an infected node can recover . δ is the probability that an infected node transforms to a susceptible state . Once recovered , the nodes could not pass the virus to other susceptible nodes , but the consequences are that the node can be infected by the same virus.

$S \rightarrow I$: In the face of the attack strategy , if the defense strategy fails, the node is infected by the attacker.At this time the attacker can exploit this node to conduct broader attacks on neighboring nodes. In this case , the attacker can use this node to attack the adjacent node .
$S \rightarrow R$: In our model , The susceptible nodes have some probability β to be immune to disease . In this case , the user decides to change the password of the account.

3.3 Application to security :

3.3.1 Implementation of cyber_attacks:

In the previous section , we described two types of random network models (erdos_renyi random, Small_World), and how to create a graph network using python. In the current section , we assure the network of size N is given to us, generated from a model described in the previous section .we are going to simulate the different modes of propagation of attacks that can help the attackers achieve their goals. we are going to see how an attacker can propagate within the network .

In cyber security, The attacker often has a specific target in mind or to simply gain control of as many internal hosts as possible , but succeeds only if the target is reached before the malicious user is detected .These two objectives used by the the attacker to avoid being detected : greater spread of an infection as much as possible to decrease the time of reaching the target, and maximizing the number of infected nodes in the network .

In our model , we consider all information about the network is completely unknown to both players . The attacker and the defender are playing at the same time , the both players choose their action without knowledge of the actions chosen by other players. We model the disease transmission according to the network_based SIR model.

At the stage ($t = 0$) , with a function ,a small number of the nodes and initially infected and nodes initially resistant are selected by the user and the rest of the population/network is susceptible. At state ($t = k$), the nodes who are infectious infect each susceptible neighbor in G , the susceptible nodes recover at a rate δ, with probability δ the password was modified to strong_password. Thus , an infected node has a probability β of recovering to become susceptible in each step while it is infected.Those who become infected by at least one infective become infectious in generation $k + 1$.

The epidemic goes on until the first generation T at which no new infections arise and all the susceptible recover to become resistant at this point the epidemic stops.

We represent the states of the population at time t with an s(t), where

$$s_t(i) = \begin{cases} 0, & \text{if node } i \text{ is susceptible} \\ 1, & \text{if node } i \text{ is infected} \\ -1, & \text{if node } i \text{ is recovered} \end{cases}$$

We consider that the nodes visualize the transformation of the game from the decision making by both players . In order to visualize decision_making , We worked with color to indicate the Current state of the node. Once the node has caught the disease, The node background color changes to red , to black if the node becomes resistant.

In our scénario , the attacker can choose between two different strategies to propagate the disease on the network . In the Strategy Based on Random , the attacker randomly chooses one or more susceptible targets among the neighbors of the infected node. In the strategy based on smart mode , the attacker, based on the node degree value, selects the node with greater degree among the neighbors of the infected node.

Random Attack:

The attacker can choose to randomly attack one or more nodes at the same time.

In order to propagate to other nodes in the network , the attacker attacks the nodes randomly. There are three methods to choose a target :unicast , multicast , broadcast .For all three methods

Unicast Method : .

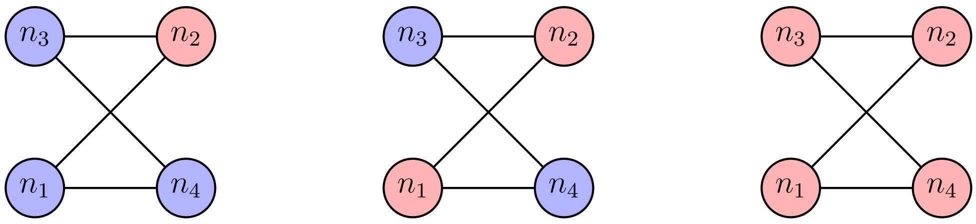
In this method , the attacker chooses to infect one one node from all their susceptible neighbors.

Here we demonstrate with algorithme how to transmit the attack on the network.

Algorithm 1: An algorithm Unicast Model

```
Data: $nodes = G.nodes$
; /* with G.nodes , we can geting a list of all nodes in the graph */
Data: $State = [0 * len(nodes)]$
; /* The list State used to represent the state of each node .All nodes are initially susceptible */
Data: $taill = G.nodes$
for $n$ to $G.nodes$ do
    $G.nodes[n]['visited'] \leftarrow False$ ; /* initially, no nodes are processed */
end
As long as there are nodes no resistance do, the boucle while continues to run
while $(State \neq 0)\&(State \neq 1)$ do
    for node to range(len(State)) do
        if $State[node]$ == 1 then
            $positions \leftarrow G[node]['index']$ ; /* returns index position of nodes in list */
            $liste\_neighbor \leftarrow list\_neighbor[position][:]$ ; /* return all neighbors of node 'node'. */
            $choix \leftarrow random.choice(liste\_neighbor)$ ; /* return a random node from the list 'choix'. */
            for $n$ to $G.nodes$ do
                if $n ==$ choix then
                    if $G.nodes[n]['visited'] \leftarrow False$ then
                        $G.nodes[n]['visited'] \leftarrow True$ ; /* mark neighbor as seen */
                        $infected\_node \leftarrow n$ ; /* add to list of new infected node */
                        $edge \leftarrow G.[nodes][n]['index']$
            end
    end
    process(push_node) ; /* to change the selected Node Color */
    process(push_edge) ; /* to change the selected edge Color */
end
```

At time $t = 0$, n_2 is the only infected node.With model unicast , n_2 has to infect only one susceptible neighbor , using random_choices() method , we randomly select a node from a list of nodes neighbors of the nodes n_2. Assuming that this attempt succeeded, at time $t = 1$, we have two infected nodes n_2 , n_1. Now , n_2 and n_1 have the opportunity to infect one or of their susceptible neighbors at the same time .That means at the time $t = 2$,we are going to have four infected nodes . in this example , we only describe the result of the algorithm unicats , the scénario of the attack unicast .



Multicast Method : .

In this strategy of propagation, the infected node randomly infects a fixed proportion of susceptible neighbors.

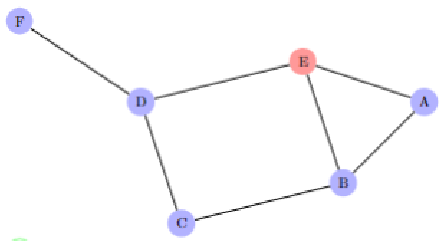
This strategy is based on the probability selected by the attacker to infect the susceptible node , calculate the number of neighbor susceptible nodes to infect .

Figure 3 demonstrates the idea of propagating multicast . For that purpose, we use again the same network from Fig. 1. We assume that the only infected node at time $t = 0$ is E.

The node E , at time $t = 0$, has a three neighbor susceptible (D, B, A) , we consider a probability $p = 0.5$.

when we calculate the number of susceptible neighbors , we multiply with the probabilities . Since the result can be float, we use the method $math.ceil(x)$ to return an Integral value.

With this probability , the attacker chooses randomly two neighbor susceptible nodes . Assuming that this attempt succeeded, at time t=1 (Fig. 2b) we have three infected nodes E , D and A. Now, E,A and F have the opportunity to infect one or more of their susceptible neighbors.

[A] 

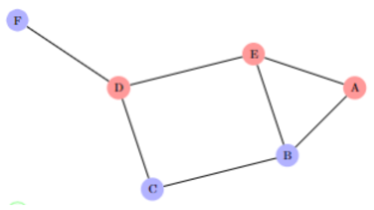
[B] 

Figure 3.1: Small-world network graphs

Here we demonstrate with algorithme how to transmit the attack on the network usint the strategy of propagation multicast ,

The first block of code in the main loop is to change all nodes to susceptible nodes ,we use the same processus as the first algorithm.

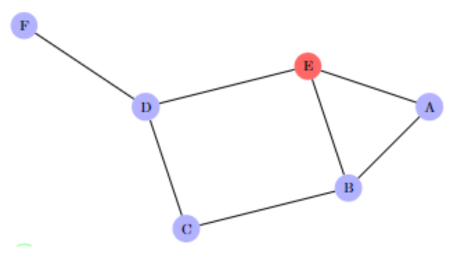
Algorithm 2: An algorithm Multicast Model

```
Initialement , same procedure as the first algorithm.
while $(State \neq 0)\&(State \neq 1)$ do
    for node to range(len(State)) do
        if $State[node]$ == 1 then
            $positions \leftarrow G[node]['index']$
            $liste\_neighbor \leftarrow list\_neighbor[position][:]$
            $\_node\_infect \leftarrow math.ceil((len(liste\_neighbor) * p)$
            $list\_choix \leftarrow random.simple(liste\_neighbor, k = nb\_node\_infect)$
            for $n$ to $G.nodes$ do
                if $G.nodes[n]['visited'] \leftarrow False$ then
                    for $j$ to $list\_choix$ do
                        if $n ==$ choix then
                            $G.nodes[n]['visited'] \leftarrow True$
                            $infected\_node \leftarrow n$
                            $edge \leftarrow G.[nodes][n]['index']$
                    end
            end
    end
    process(push_node(infected_node) )
    process(push_edge(edg))
end
```

Broadcast Method : .

In this strategy of propagation, the infected nodes infect all their susceptible neighbor.

Figure 3.2 demonstrates the contagion process according to the broadcast propagation strategy. For that purpose, we again use the same network from Fig 3.1 , At time t=0 (Fig. 3.2A), E is the only infected node. E has a three neighbor susceptible (D, B, A). In this strategy , the infected nodes infect globally all their susceptible neighbors. at time =1(Fig. 3.2B), , we have four infected nodes E , ,B, D and A.

[A] 

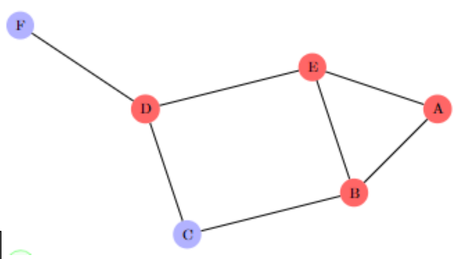
[B] 

Figure 3.2: Small-world network graphs

Here we demonstrate with algorithme how to transmit the attack on the network using the strategy of propagation broadcast .

Algorithm 3: An algorithm Broadcast Model

```
Initialement , same procedure as the first algorithm.
while $(State \neq 0)\&(State \neq 1)$ do
    for node to range(len(State)) do
        if $State[node]$ == 1 then
            $positions \leftarrow G[node]['index']$
            $liste\_neighbor \leftarrow list\_neighbor[position][:]$
            for n to G.nodes do
                for j to liste_neighbor do
                    if $G.nodes[n]['visited'] \leftarrow False$ then
                        if $n ==$ j then
                            $G.nodes[n]['visited'] \leftarrow True$
                            $infected\_node \leftarrow n$
                            $edge \leftarrow G.[nodes][n]['index']$
                end
            end
    end
    process(push_node(infected_node) )
    process(push_edge(edg))
end
```

Smart Attack:

In this model , we introduit a new strategy of attack select nodes with the highest degree . The model is characterized by the degree of centrality . The degree centrality of nodes is simply the number of edges it has. The higher the degree, the more central the node is .

There are three methods to choose a target :deterministic_smart , probabilistic_smart , broadcast_smart .For all three methods .

....

deterministic_smart

In this method , the attacker chooses to infect one node with the high degree .

suppose that at time t=0 , the node E is the only infected node , and the node E has three susceptible neighbors (C,D,A). Using the smart strategy , the attackers select one of the nodes' neighbors with high degree.In this exemple , in Figure 21.5, both nodes D and B have the same high degree , and both nodes have the same probability to be chosen, so the attackers randomly select one among the two nodes with high degree .

Here we demonstrate with algorithme how to transmit the attack on the network.

broadcast_smart

In this method , the attacker chooses to infect all node with the high degree .

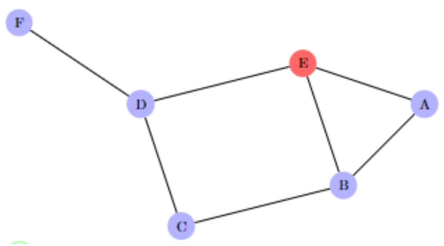
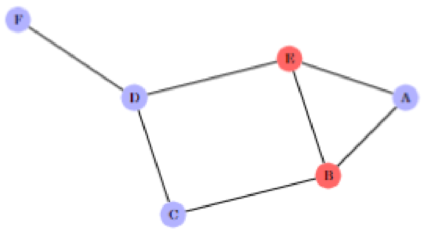
[A]  [B] 

Figure 3.3: Small-world network graphs

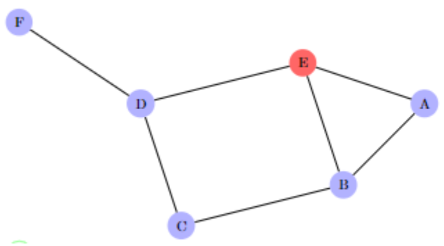
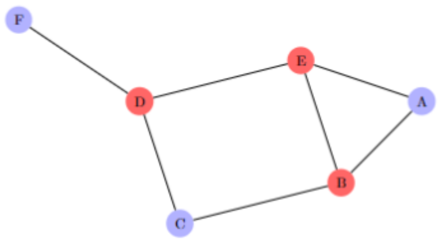
[A]  [B] 

Figure 3.4: Small-world network graphs

3.3.2 Implementation of cyber_defense mechanisms :

In traditional cyber defense scenarios, the defender waits until an attacker makes a move and then responds. This strategy results in a situation in which the attacker has the advantage and the defender must clean up and deal with the repercussions of the attack. In our cyber defense scenarios, we attempt to create a situation in which the defender takes an action against an attacker by active measures of detection , protection and response .

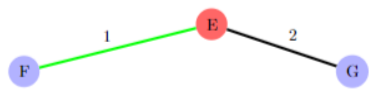
A Honeypot is a type of cyber-defense used against cyber-attacks to slow down an attacker and make cyber-attacks more difficult to carry out.

When attackers find a vulnerability on a network, they often list targets within the network for services they might be able to explore.By running a honeypot on the perimeter of our network, we can monitor its activity. Hopefully we can catch attackers early on in the exploit and react before any real damage is done.
A complete honeypot system that we will demonstrate in this article has two essential modules: decoys and security programmes . The decoy is represented by Intrusion Detection System (IDS) for detected suspicious activities and generates alerts when they are detected, and the security programme includes the security functions, Based on the source of these alerts , the honeypot takes the appropriate action to deal with the threat after analyzing the issue.

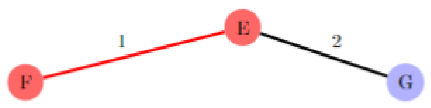
This article aims to specify a model of honeypots attack. Our model is completely different to other models of honeypot; in our paper , we suggest that the defender places the honeypot on the edge, not on the nodes . The defender has to decide which edge to allocate the honeypot on.

To understand attacker and defender motives, strategies, we introduit an example scenario with the actors the attacker and defender actions . we consider a network (fig. . .) .In this scenario, we assume three nodes , E, F, G , connected through two edges. the attacker currently entering the network through node E . The attacker has to decide whether to attack the node F or G . On the other side , the defender decides to allocate one honeypot to the existing edges between E and F or the other edge. We consider two situations .

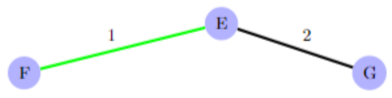
The first situation ;if the attacker decides to attack the node F, at the same time , the defender allocates the honeypot to the edges between E , F . In this case , the attacker uses the same edge used by the defender , and the defender detects the intrusions , and after he is going to respond by recovering the two nodes E and F .



[A]



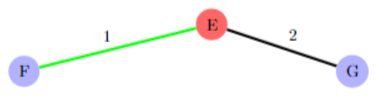
[B]



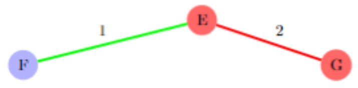
[C]

Figure 3.5: Small-world network graphs

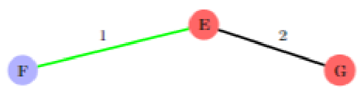
The second situation , the attacker chooses to infect the node G , at the same time , The defender chooses to block the edges between E , F . The defense strategy fails, the node G is penetrated by the attacker.



[A]



[B]



[C]

Figure 3.6: Small-world network graphs

There are two different ways to place a honeypot in the network: random or smart ways:

The random strategy :the system defense generates a random edge to place a honeypot . The only condition is that the edge connects two nodes that are not resistant.

The smart strategy : when a honeypot detects suspicious activities from a node , the system defense places the honeypot between the source of node and its neighbors.

We use the algorithm to obtain the strategie to allocate honeypots over an attack graph.We represent the body of the function with four blocks :

When the defender detects suspicious activities between two nodes , we select the source of the suspicious activities as the node_out and the destination of the action as the node_in.

The first block of code is to return all the neighbors of node_out except the resistance nodes and the node_in, The second block of code to get the data for the edges between the node_out and her neighbors . The third block of code to return all the active edges in the network. the edge in active mode,If an edge is not connected to any resistant node .

....

Algorithm 4: An algorithm defence Model

```
for $n$ to $node\_out$ do
    $list\_neigh\_out.remove(list\_node\_resist, node\_in)$ ;      /* return the
      neighbors of the node_ont , and removing the resistance
      node and node_in  */
end
for $k$ to $list\_neigh$ do
    for $j$ to $len(list\_neigh[k][:])$ do
        $edge\_smart \leftarrow G[k][j]['index']$ ;      /* return the data for the
          edges between the node_out and her neighbors (list_neigh)
          .    */
    end
end
for $n$ to $G.nodes$ do
    for $i$ to $list\_neigh\_normal$ do
        $edge\_normal \leftarrow G[i][n]['index']$;
          $edge\_normal.remove(edge\_smart, list\_edge\_resist);)$; ;   /* return
          the data for the active edges  */
    end
end
```

And the finally block of code allocated randomly the honeypot between two nodes .

we begin

Chapter 4

Conclusion

Acknowledgment

References

[1] Write the reference here