**Practical-1**

**Aim: Perform a practical to demonstrate ping of death (Denial of Service) attack in Ubuntu machine.**

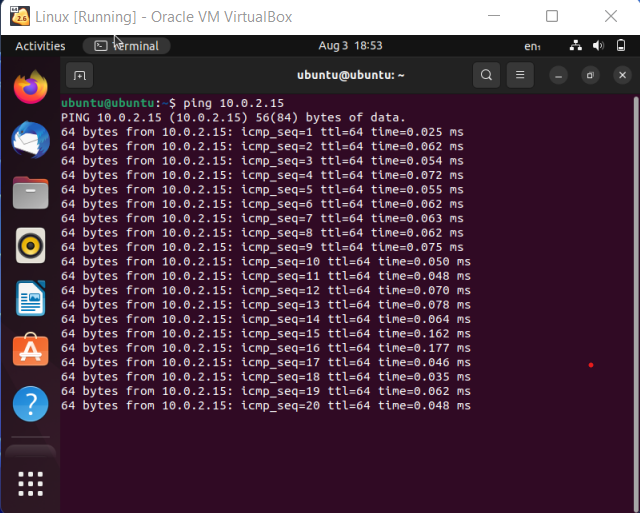
**Denial Of service:-**

A Denial-of-Service (DoS) attack is **an attack meant to shut down a machine or network, making it inaccessible to its intended users**. DoS attacks accomplish this

**Ping Command:**

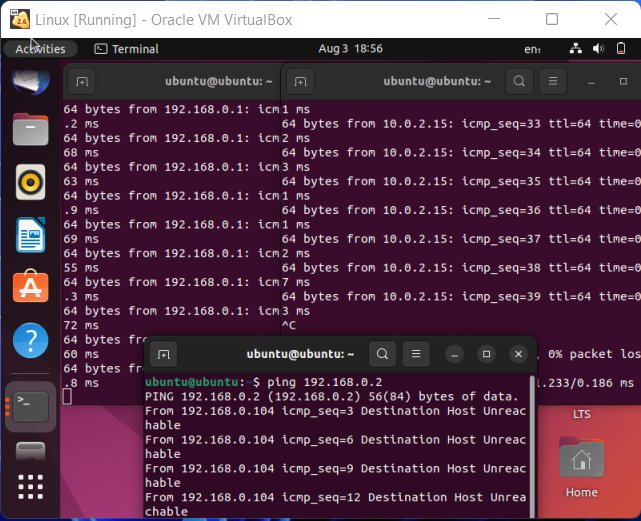
**PING (Packet Internet Groper)** command is used to check the network connectivity between host and server/host.

* we use ping 10.0.2.15



* Ping Command To Ping Multiple IP address.

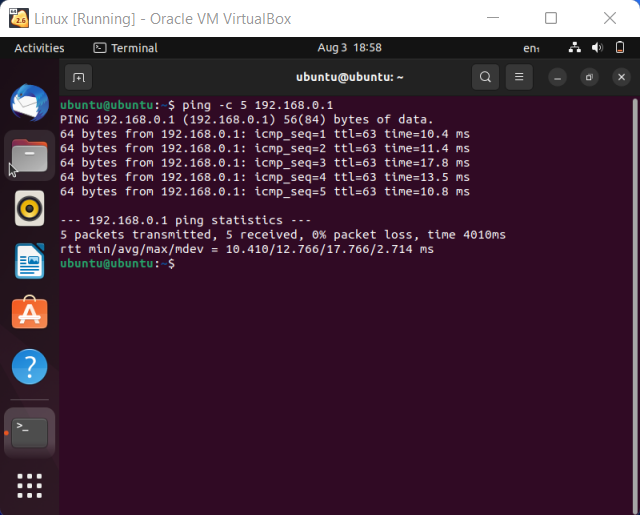
**Command:** ping 192.168.0.1 ,ping 192.168.0.2,ping 192.168.0.3



* **Ping Command Options:**

1. **-c [count]**

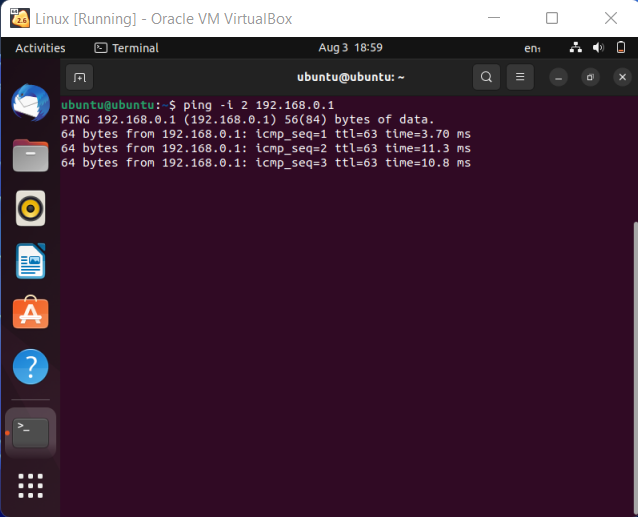
**Use:** Stop after sending count ECHO\_REQUEST packets. With deadline option, ping waits for count ECHO\_REPLY packets, until the time‐ out expires.

**Output:**

1. **-i [interval]:**

**Use**: Wait interval seconds between sending each packet. The default is to wait for one second between each packet normally, or not to wait in flood mode. Only super-user may set interval to values less 0.2 seconds.

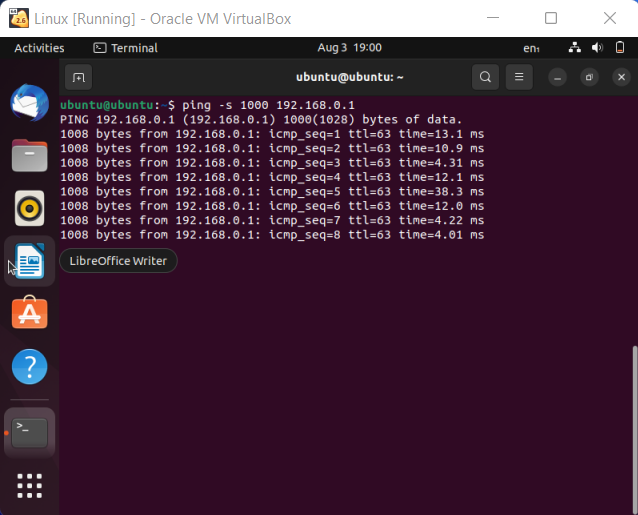
**Output:**



1. **-s [size]:**

**Use**: Specifies the number of data bytes to be sent. The default is56, which translates into 64 ICMP data bytes when combined with the 8 bytes of ICMP header data.

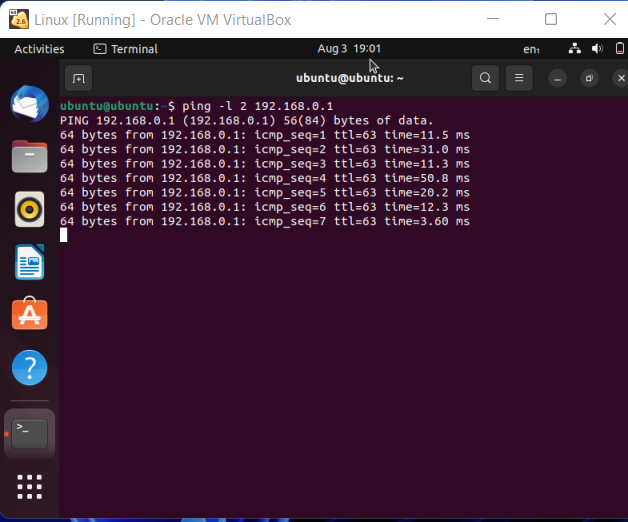
**Output**:



1. **-l [preload]:**

**Use:** If preload is specified, ping sends that many packets not waiting for reply. Only the super-user may select preload more than one.

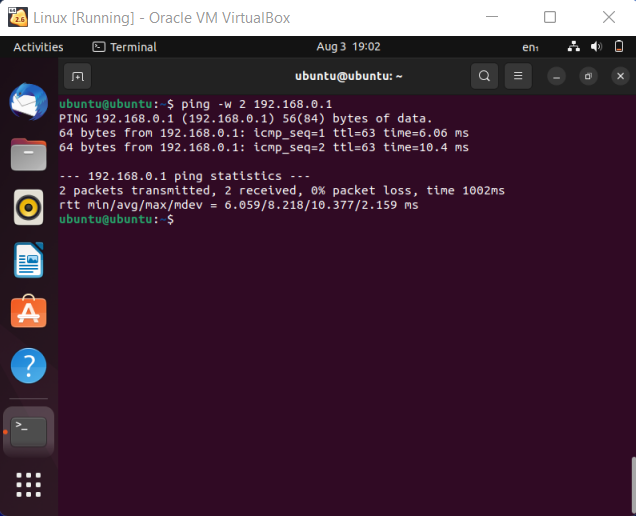
**Output**:



1. **-w [deadline]:**

**Use:** Specify a timeout, in seconds, before ping exits regardless ofhow many packets have been sent or received. In this case ping does not stop after count packet are sent, it waits either for deadline expire or until count probes are answered or for some error notification from network.

**Output:**



**Practical-2**

**Aim: Perform a practical to install network mapper tool and analyze the open ports in your Ubuntu machine.** **Perform following using NMAP.**

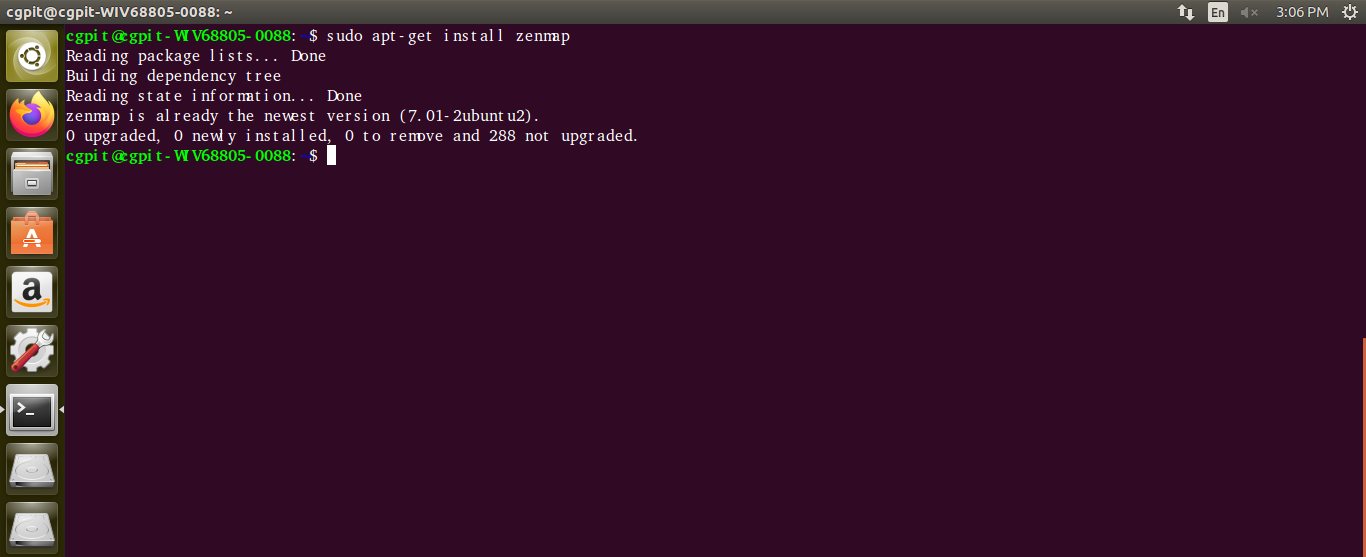
**a.Find open ports on a system**

**b. Find the machines which are active**

**c. Find the version of remote OS on other systems Find the version of Software installed on other system**

* **Installation of Network Mapping Tool.**

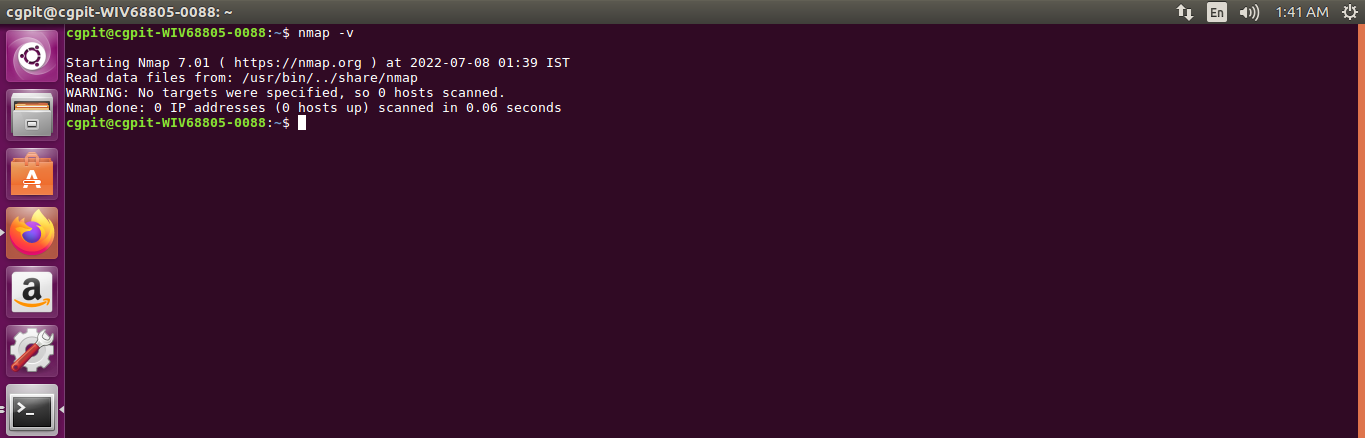
Zenmap is **the official Nmap Security Scanner GUI**. It is a multi-platform (Linux, Windows, Mac OS X, BSD, etc.) free and open source application which aims to make Nmap easy for beginners to use while providing advanced features for experienced Nmap users.



* **Command** Of **Nmap** Using **Terminal** And **GUI**.

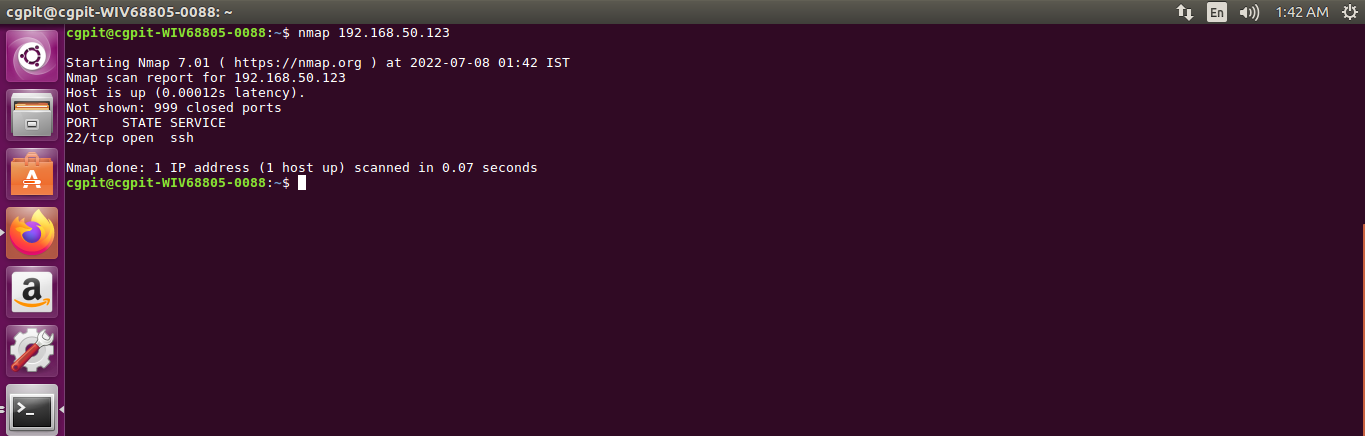
1. **Nmap -v**

* It is Used To Check Version of Nmap



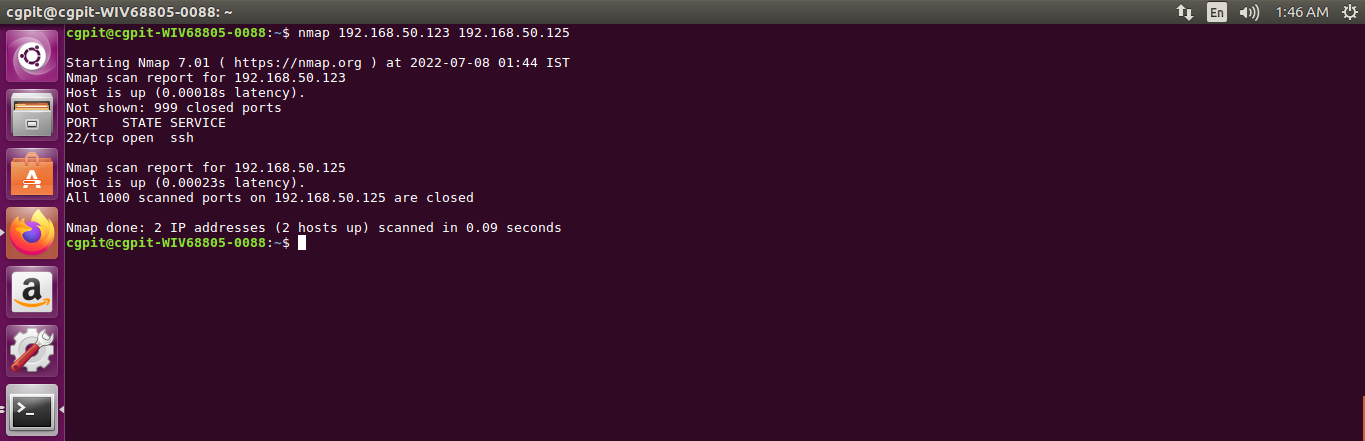
1. **Nmap 192.168.50.123**

* Scan a Single Target



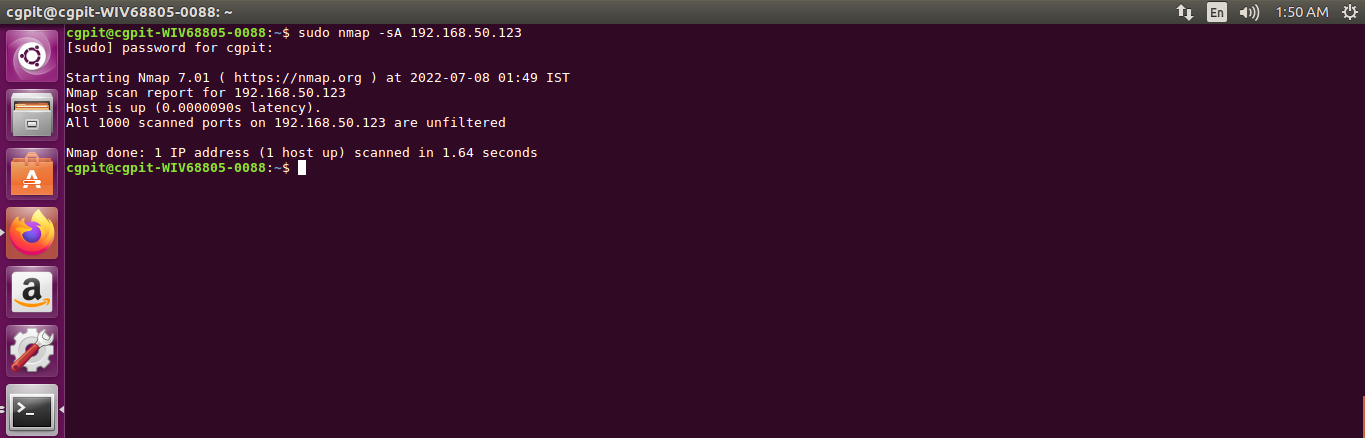
1. **Nmap 192.168.50.123 192.168.50.125**

* Scan Multiple Targets



1. **sudo nmap -sA 192.168.50.123**

* To Scan System Is protected By Firewall or Not.



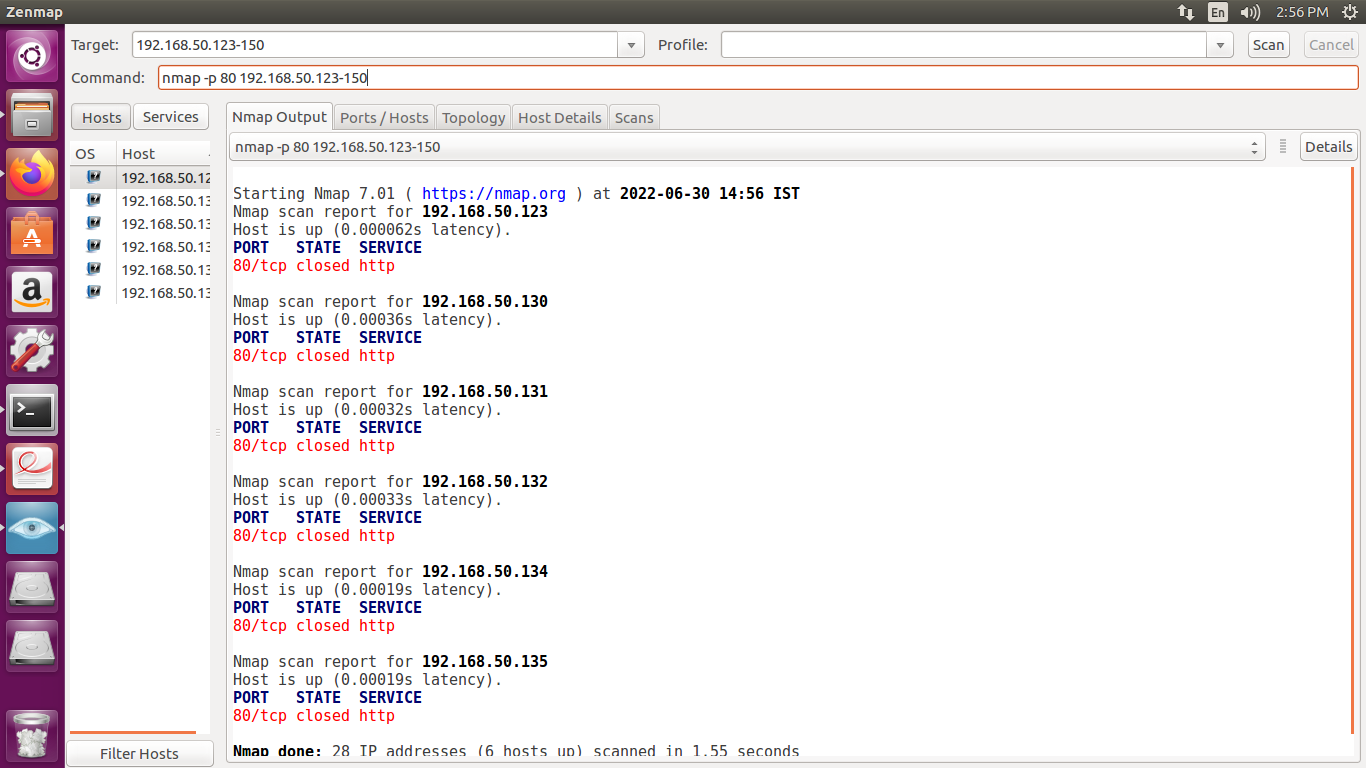
1. **Nmap -p 80 192.168.50.123**

* To Scan Specific Port Is Open or Not.



1. **Nmap -p 80 192.168.50.123-150**

* To Scan specific Port is open or not with IP Address Range.



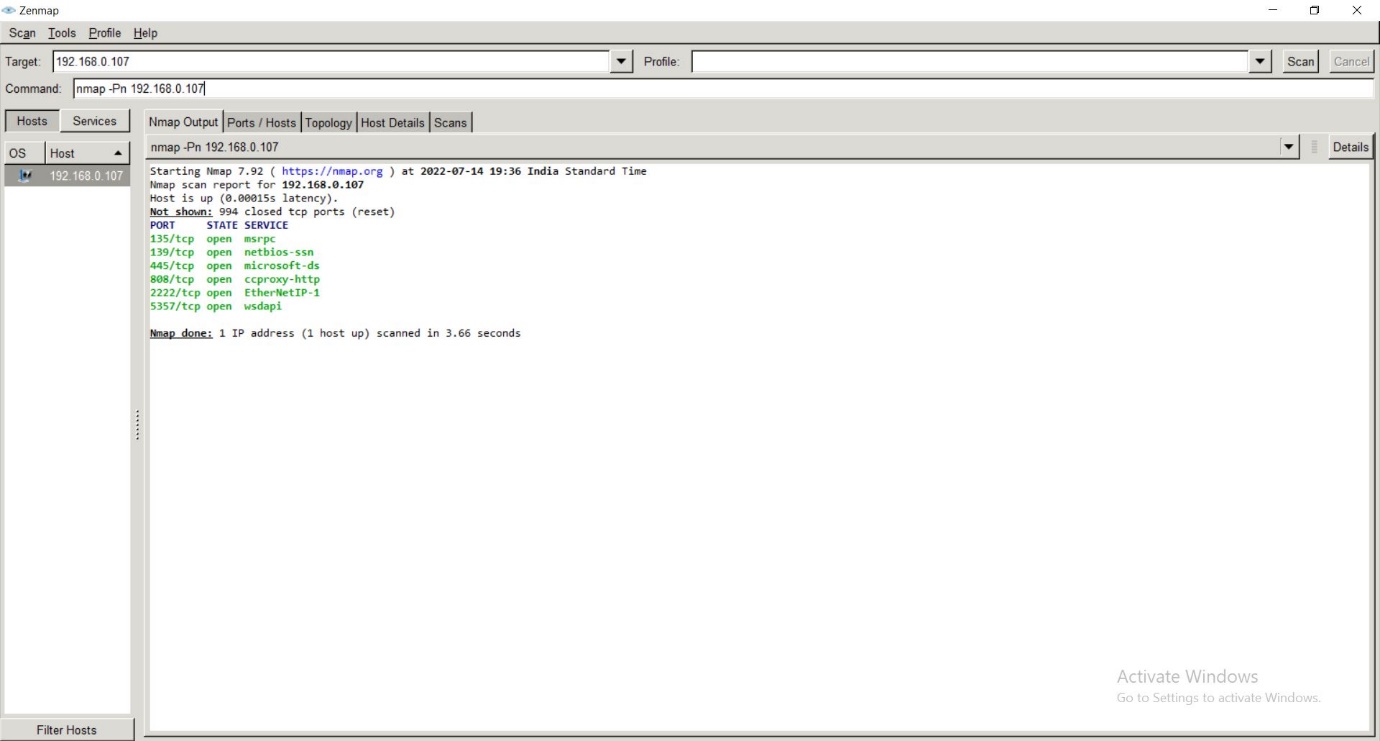
1. **Namp -sL 192.168.50.123**

* Create a Host List



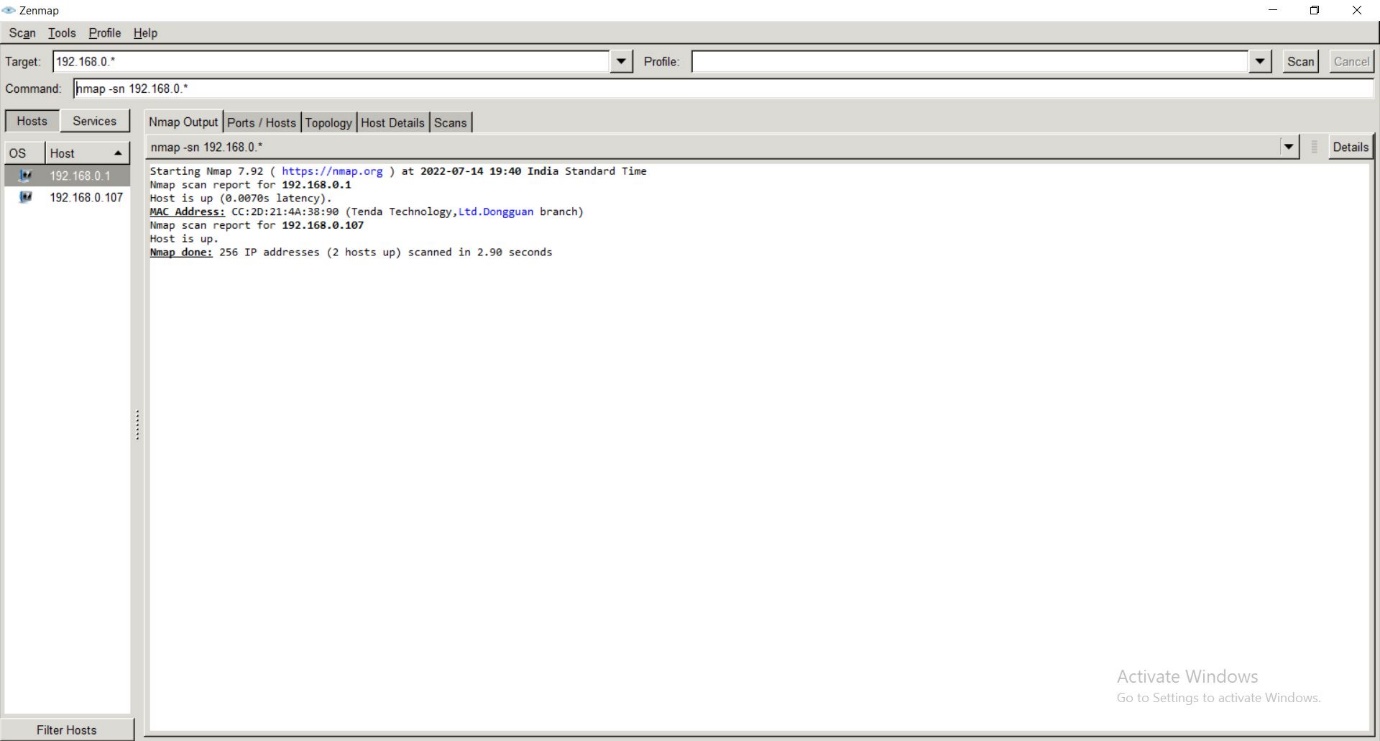
1. **Nmap -Pn 192.168.0.107**

* Disable host discovery. Port scan only.



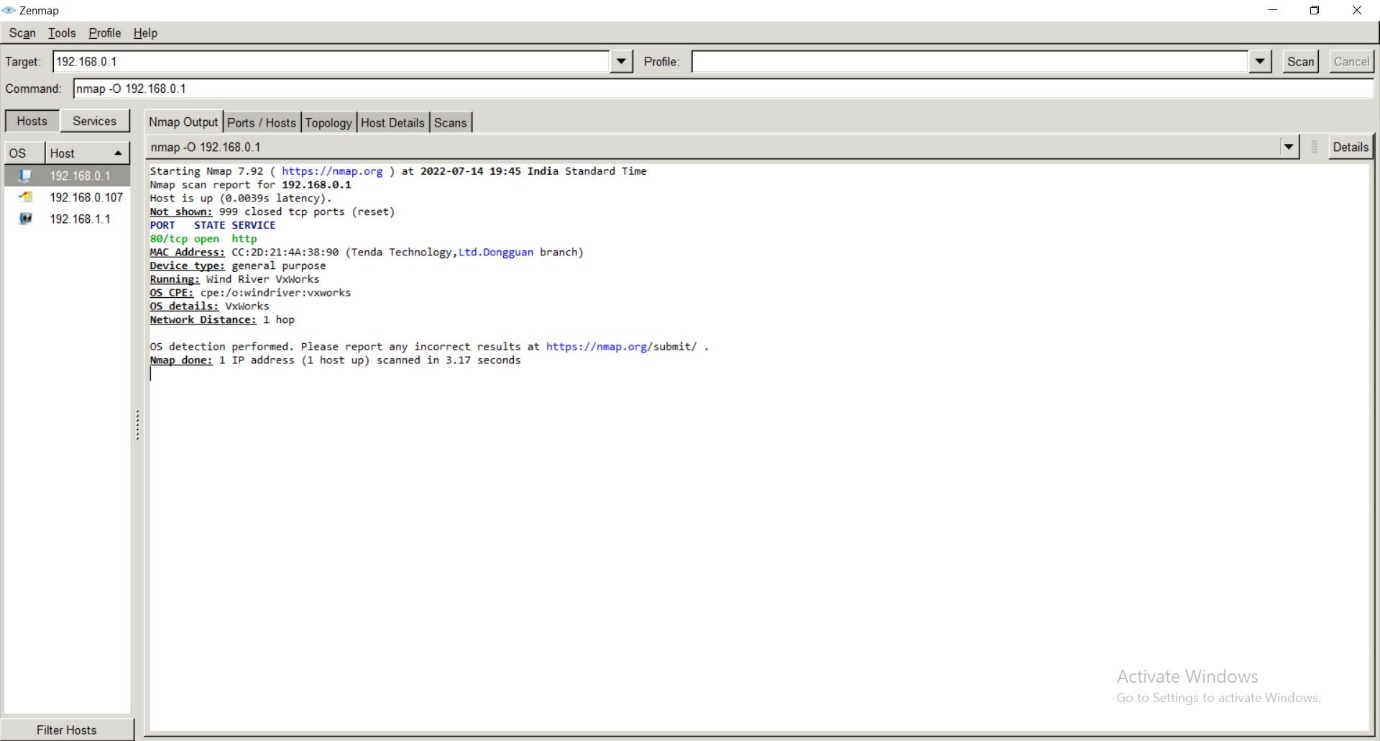
1. **Nmap -sn 192.168.0.\***

* Disable port scanning. Host discovery only.



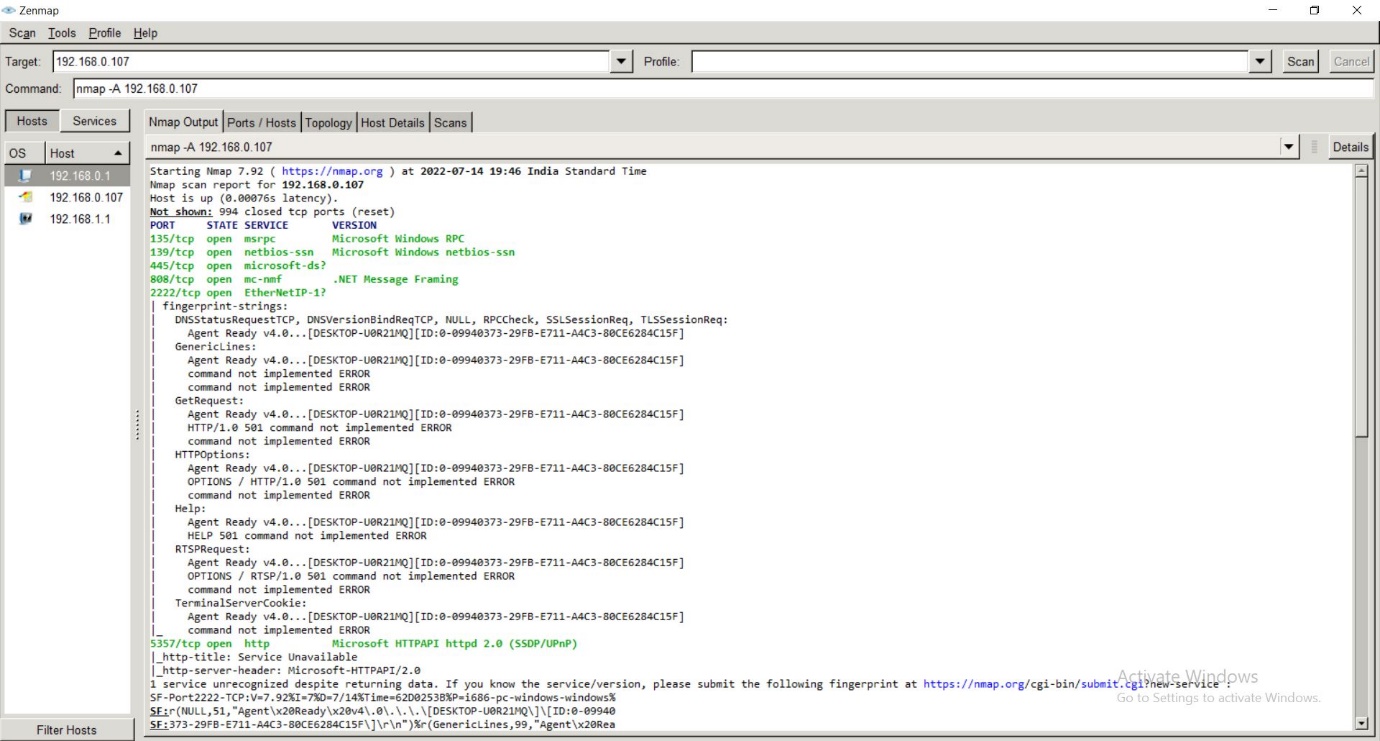
1. **Nmap -O 192.168.0.1**

* Remote OS detection using TCP/IP stack fingerprinting



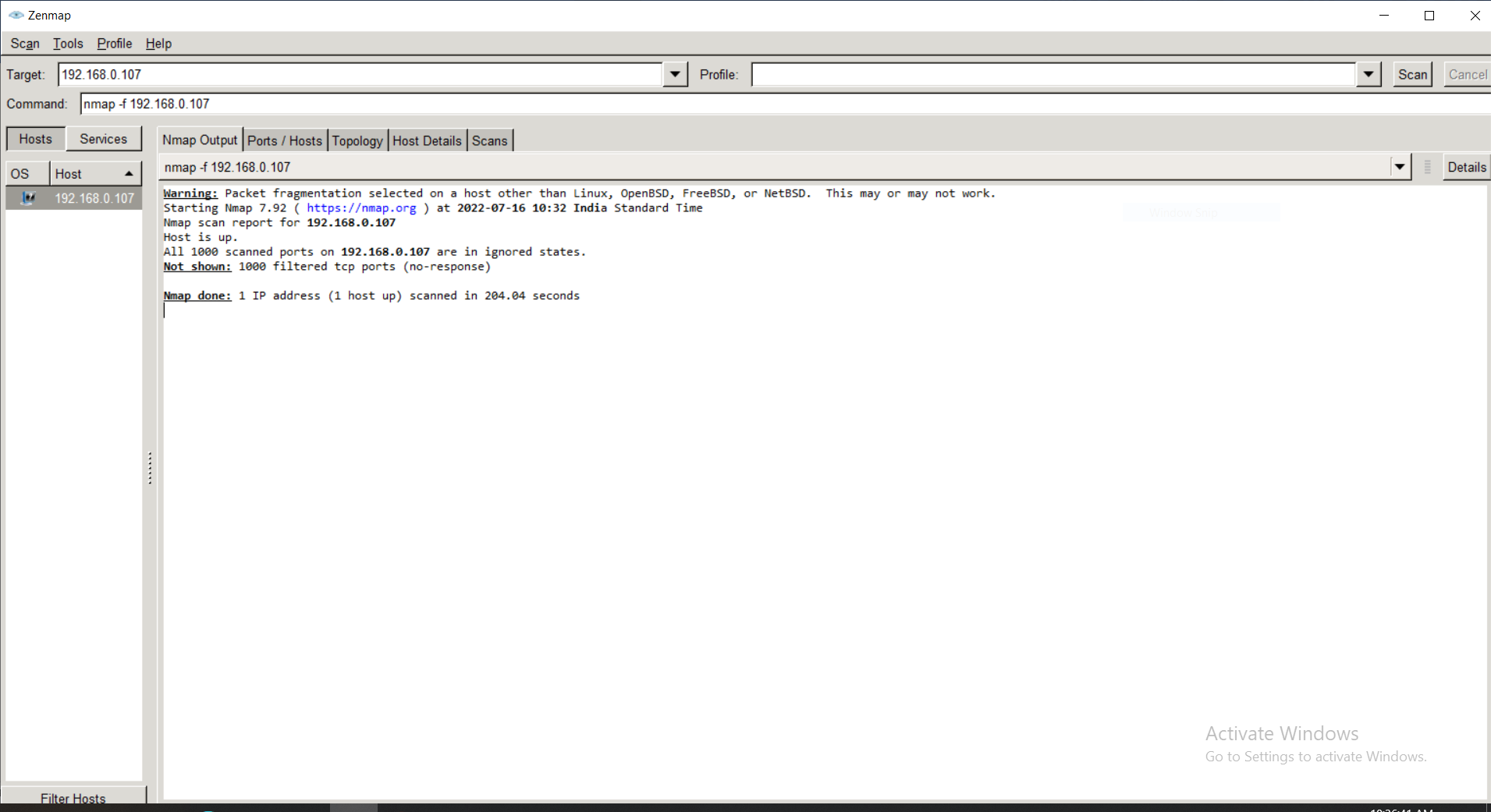
1. **Nmap -A 192.168.0.107**

* Enables OS detection, version detection, script scanning, and traceroute



1. **Nmap -f 192.168.0.107**

* augment Packets



**Practical-3**

**Aim:** **Perform a practical to implement Caesar cipher and play fair cipher.**

**Caesar Cipher:**

**Code:**

# we need 2 helper mappings, from letters to ints and the inverse

L2I = dict(zip("ABCDEFGHIJKLMNOPQRSTUVWXYZ",range(26)))

I2L = dict(zip(range(26),"ABCDEFGHIJKLMNOPQRSTUVWXYZ"))

key =int(input("Enter Key:"))

plaintext =input("Enter Plaintext:")

# encryption

def encryption():

ciphertext = ""

for c in plaintext.upper():

if c.isalpha(): ciphertext += I2L[ (L2I[c] + key)%26 ]

else: ciphertext += c

return ciphertext

# decryption

def decryption():

ciphertext=input("Enter CipherText:")

plaintext2 = ""

for c in ciphertext.upper():

if c.isalpha(): plaintext2 += I2L[ (L2I[c] - key)%26 ]

else: plaintext2 += c

return plaintext2

while(1):

c=int(input("-------Enter Your Choice-------\n1.Encryption\n2.Decryption\n3.Exit\nChoice:"))

if c==1:

print("cipherTexte:",encryption())

elif c==2:

print("PlainText:",decryption())

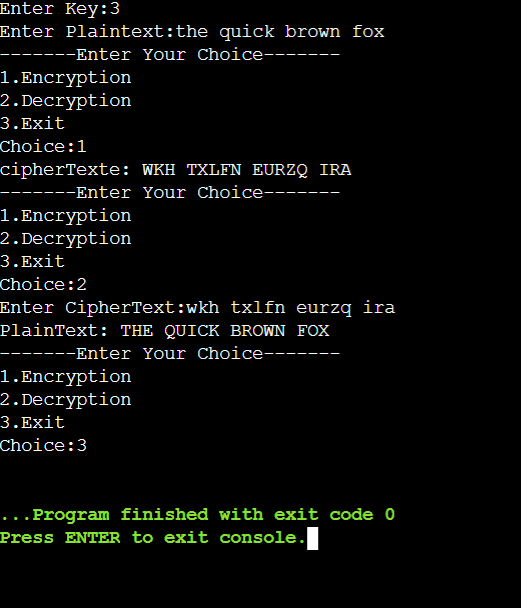
elif c==3:

exit()

else:

print("Choose Correct Choice!!!")

**Output:**



**Playfair cipher:**

**Code:**

key=input("Enter key:")

key=key.replace(" ", "")

key=key.upper()

def matrix(x,y,initial):

return [[initial for i in range(x)] for j in range(y)]

result=list()

for c in key: #storing key

if c not in result:

if c=='J':

result.append('I')

else:

result.append(c)

flag=0

for i in range(65,91): #storing other character

if chr(i) not in result:

if i==73 and chr(74) not in result:

result.append("I")

flag=1

elif flag==0 and i==73 or i==74:

pass

else:

result.append(chr(i))

k=0

my\_matrix=matrix(5,5,0) #initialize matrix

for i in range(0,5): #making matrix

for j in range(0,5):

my\_matrix[i][j]=result[k]

k+=1

def locindex(c): #get location of each character

loc=list()

if c=='J':

c='I'

for i ,j in enumerate(my\_matrix):

for k,l in enumerate(j):

if c==l:

loc.append(i)

loc.append(k)

return loc

def encrypt(): #Encryption

msg=str(input("ENTER PLAIN TEXT:"))

msg=msg.upper()

msg=msg.replace(" ", "")

i=0

for s in range(0,len(msg)+1,2):

if s<len(msg)-1:

if msg[s]==msg[s+1]:

msg=msg[:s+1]+'X'+msg[s+1:]

if len(msg)%2!=0:

msg=msg[:]+'X'

print("CIPHER TEXT:",end=' ')

while i<len(msg):

loc=list()

loc=locindex(msg[i])

loc1=list()

loc1=locindex(msg[i+1])

if loc[1]==loc1[1]:

print("{}{}".format(my\_matrix[(loc[0]+1)%5][loc[1]],my\_matrix[(loc1[0]+1)%5][loc1[1]]),end=' ')

elif loc[0]==loc1[0]:

print("{}{}".format(my\_matrix[loc[0]][(loc[1]+1)%5],my\_matrix[loc1[0]][(loc1[1]+1)%5]),end=' ')

else:

print("{}{}".format(my\_matrix[loc[0]][loc1[1]],my\_matrix[loc1[0]][loc[1]]),end=' ')

i=i+2

print("\n")

def decrypt(): #decryption

msg=str(input("ENTER CIPHER TEXT:"))

msg=msg.upper()

msg=msg.replace(" ", "")

print("PLAIN TEXT:",end=' ')

i=0

while i<len(msg):

loc=list()

loc=locindex(msg[i])

loc1=list()

loc1=locindex(msg[i+1])

if loc[1]==loc1[1]:

print("{}{}".format(my\_matrix[(loc[0]-1)%5][loc[1]],my\_matrix[(loc1[0]-1)%5][loc1[1]]),end=' ')

elif loc[0]==loc1[0]:

print("{}{}".format(my\_matrix[loc[0]][(loc[1]-1)%5],my\_matrix[loc1[0]][(loc1[1]-1)%5]),end=' ')

else:

print("{}{}".format(my\_matrix[loc[0]][loc1[1]],my\_matrix[loc1[0]][loc[1]]),end=' ')

i=i+2

print("\n")

while(1):

choice=int(input("-------Enter Choice-------\n1.Encryption\n2.Decryption\n3.EXIT\nchoice:"))

if choice==1:

encrypt()

elif choice==2:

decrypt()

elif choice==3:

exit()

else:

print("Choose Correct Choice!!!")

**Output:**

