**PRACTICAL 1**

**AIM: Implementing Substitution and Transposition Ciphers.**

Design and implement algorithms to encrypt and decrypt messages using classical substitution and transposition techniques.

**Program 1:** Write a python program to implement Ceaser Cipher.

def encrypt(string, shift):

cipher = ''

for char in string:

if char == ' ':

cipher = cipher + char

elif char.isupper():

cipher = cipher + chr((ord(char) + shift - 65) % 26 + 65)

else:

cipher = cipher + chr((ord(char) + shift - 97) % 26 + 97)

return cipher

def decrypt(string, shift):

cipher = ''

for char in string:

if char == ' ':

cipher = cipher + char

elif char.isupper():

cipher = cipher + chr((ord(char) + (26-shift) - 65) % 26 + 65)

else:

cipher = cipher + chr((ord(char) + (26-shift) - 97) % 26 + 97)

return cipher

text = input("Enter String : ")

s = int(input("enter Shift Number : "))

option = int(input("1. For Encrypt \n2. For Decrypt\n Enter Your choice : "))

print("Original String : ", text)

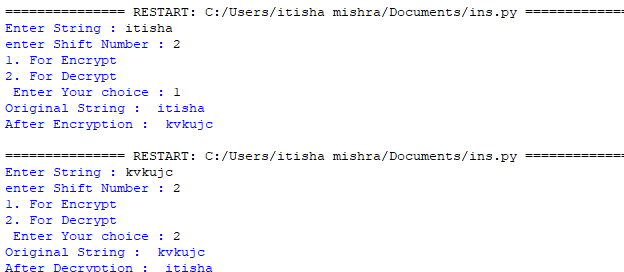
if( option == 1):

print("After Encryption : ", encrypt(text, s))

else:

print("After Decryption : ", decrypt(text, s))

**Output:**



**Program 2:** Write a python program to implement Mono-alphabetic Cipher.

alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

keyword = "ZYXWVUTSRQPONMLKJIHGFEDCBA"

def encrypt(Plaintext):

result = ""

for char in Plaintext:

if char in alphabet:

num = alphabet.find(char)

result += keyword[num]

else:

result += char

print("Encrypted Text:", result)

def decrypt(Ciphertext):

result = ""

for char in Ciphertext:

if char in keyword:

num = keyword.find(char)

result += alphabet[num]

else:

result += char

print("Decrypted Text:", result)

while True:

try:

n = int(input("Enter Value:\n1) Encrypt Text\n2) Decrypt Text\n3) See Key\n4) Exit\nChoice: "))

except ValueError:

print("Invalid input; please enter a number between 1 and 4.")

continue

if n == 1:

Plaintext = input("Enter Text to Encrypt: ")

encrypt(Plaintext.upper())

elif n == 2:

Ciphertext = input("Enter Text to Decrypt: ")

decrypt(Ciphertext.upper())

elif n == 3:

print("Substitution Key (Keyword):", keyword)

elif n == 4:

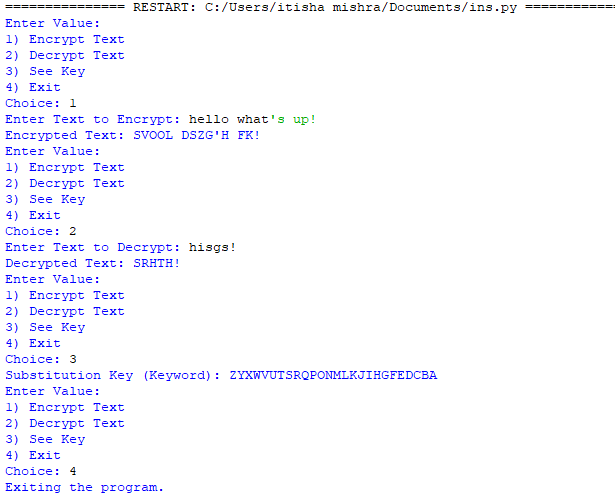
print("Exiting the program.")

break

else:

print("Invalid Input; Enter Again!!")

**Output:**



**Program 3:** Write a python program to implement Playfair Cipher.

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:

if c not in result:

if c == 'J':

result.append('I')

else:

result.append(c)

flag = 0

for i in range(65, 91):

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)

for i in range(0, 5):

for j in range(0, 5):

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

k += 1

def locindex(c):

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():

msg = str(input("ENTER MSG : "))

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

def decrypt():

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

while (1):

choice = int(input("\n 1.Encryption \n 2.Decryption: \n 3.EXIT \n Enter Your Choice: \n "))

if choice == 1:

encrypt()

elif choice == 2:

decrypt()

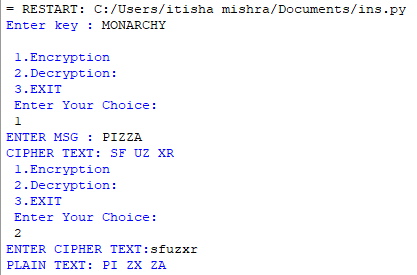
elif choice == 3:

exit()

else:

print("Choose correct choice")

**Output:**



**Program 4:** Write a python program to implement Vernam Cipher.

def Vernam(Plain,Key,Flag):

result=""

for i in range(len(Plain)):

char=Plain[i]

if(Flag):

result+=chr((ord(char)-97 +ord(Key[i])-97)%26 +97)

else:

result += chr((ord(char) - ord(Key[i])+26) % 26 + 97)

return result

if \_\_name\_\_=="\_\_main\_\_":

Key=''.join(input("Enter Key: ").lower().split())

Plain=''.join(input("Enter Plaintext: ").lower().split())

if(len(Key)!=len(Plain)):

print("Invalid Key!")

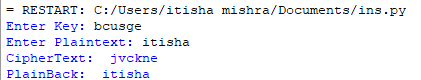
exit(None)

CipherText=Vernam(Plain,Key,True)

print("CipherText: ",CipherText)

print("PlainBack: ",Vernam(CipherText,Key,False))

**Output:**

****

**Program 5:** Write a python program to implement Simple Columnar Transposition Cipher.

import math

key = "HACK"

def encryptMessage(msg):

cipher = ""

k\_indx = 0

msg\_len = float(len(msg))

msg\_lst = list(msg)

key\_lst = sorted(list(key))

col = len(key)

row = int(math.ceil(msg\_len / col))

fill\_null = int((row \* col) - msg\_len)

msg\_lst.extend('\_' \* fill\_null)

matrix = [msg\_lst[i: i + col]

for i in range(0, len(msg\_lst), col)]

for \_ in range(col):

curr\_idx = key.index(key\_lst[k\_indx])

cipher += ''.join([row[curr\_idx]

for row in matrix])

k\_indx += 1

return cipher

def decryptMessage(cipher):

msg = ""

k\_indx = 0

msg\_indx = 0

msg\_len = float(len(cipher))

msg\_lst = list(cipher)

col = len(key)

row = int(math.ceil(msg\_len / col))

key\_lst = sorted(list(key))

dec\_cipher = []

for \_ in range(row):

dec\_cipher += [[None] \* col]

for \_ in range(col):

curr\_idx = key.index(key\_lst[k\_indx])

for j in range(row):

dec\_cipher[j][curr\_idx] = msg\_lst[msg\_indx]

msg\_indx += 1

k\_indx += 1

try:

msg = ''.join(sum(dec\_cipher, []))

except TypeError:

raise TypeError("This program cannot",

"handle repeating words.")

null\_count = msg.count('\_')

if null\_count > 0:

return msg[: -null\_count]

return msg

msg = "Come Home Tomorrow"

cipher = encryptMessage(msg)

print("Encrypted Message: {}".

format(cipher))

print("Decryped Message: {}".

format(decryptMessage(cipher)))

**Output:**



Program 6: Write a python program to implement Railfence Cipher.

def encryptRailFence(text, key):

rail = [['\n' for i in range(len(text))]

for j in range(key)]

dir\_down = False

row, col = 0, 0

for i in range(len(text)):

if (row == 0) or (row == key - 1):

dir\_down = not dir\_down

rail[row][col] = text[i]

col += 1

if dir\_down:

row += 1

else:

row -= 1

result = []

for i in range(key):

for j in range(len(text)):

if rail[i][j] != '\n':

result.append(rail[i][j])

return("" . join(result))

def decryptRailFence(cipher, key):

rail = [['\n' for i in range(len(cipher))]

for j in range(key)]

dir\_down = None

row, col = 0, 0

for i in range(len(cipher)):

if row == 0:

dir\_down = True

if row == key - 1:

dir\_down = False

rail[row][col] = '\*'

col += 1

if dir\_down:

row += 1

else:

row -= 1

index = 0

for i in range(key):

for j in range(len(cipher)):

if ((rail[i][j] == '\*') and

(index < len(cipher))):

rail[i][j] = cipher[index]

index += 1

result = []

row, col = 0, 0

for i in range(len(cipher)):

if row == 0:

dir\_down = True

if row == key-1:

dir\_down = False

if (rail[row][col] != '\*'):

result.append(rail[row][col])

col += 1

if dir\_down:

row += 1

else:

row -= 1

return("".join(result))

if \_\_name\_\_ == "\_\_main\_\_":

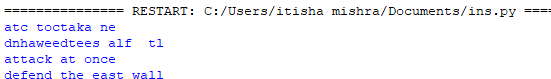
print(encryptRailFence("attack at once", 2))

print(encryptRailFence("defend the east wall", 3))

print(decryptRailFence("atc toctaka ne", 2))

print(decryptRailFence("dnhaweedtees alf tl", 3))

**Output:**



**PRACTICAL 2**

**AIM: RSA Encryption and Decryption: Implement the RSA algorithm for public-key encryption and decryption, and explore its properties and security considerations.**

**Program :**

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

import binascii

keyPair = RSA.generate(1024)

pubKey = keyPair.publickey()

print(f"Public key: (n={hex(pubKey.n)}, e={hex(pubKey.e)})")

pubKeyPEM = pubKey.export\_key()

print(pubKeyPEM.decode('utf-8'))

print(f"Private key: (n={hex(keyPair.n)}, d={hex(keyPair.d)})")

privKeyPEM = keyPair.export\_key()

print(privKeyPEM.decode('utf-8'))

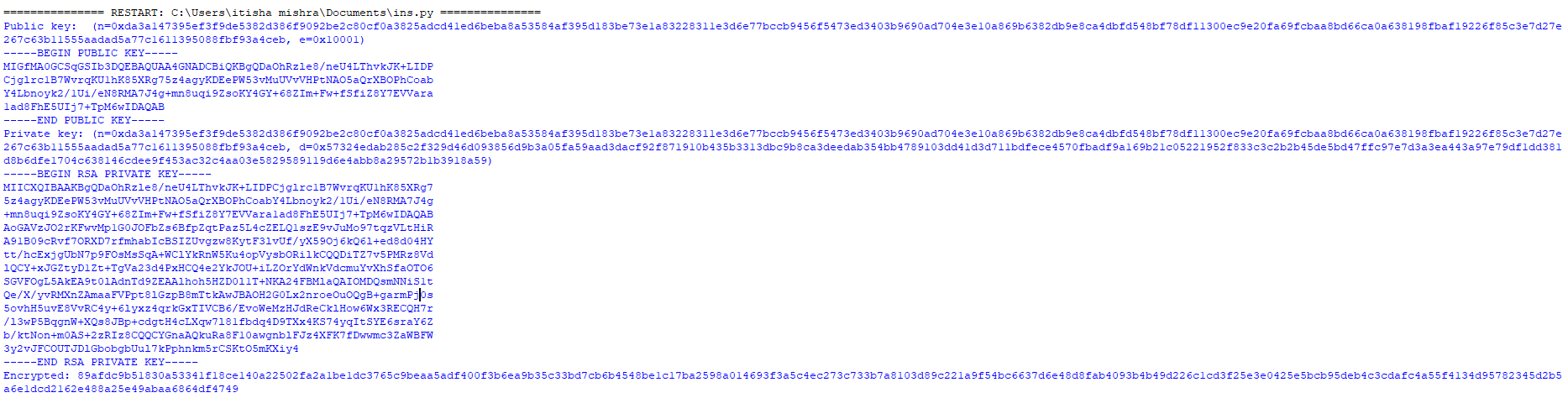
msg = b'Hello Class'

encryptor = PKCS1\_OAEP.new(pubKey)

encrypted = encryptor.encrypt(msg)

print("Encrypted:", binascii.hexlify(encrypted).decode('utf-8'))

**Output:**

****

**PRACTICAL 3**

**AIM: Message Authentication Codes: Implement algorithms to generate and verify message authentication codes (MACs) for ensuring data integrity and authenticity.**

**Program 1:**

import hashlib

result = hashlib.md5(b'Hello')

result1 = hashlib.md5(b'Fello')

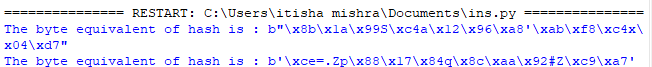
print("The byte equivalent of hash is : ", end ="")

print(result.digest())

print("The byte equivalent of hash is : ", end ="")

print(result1.digest())

**Output:**

****

**Program 2:**

import hashlib

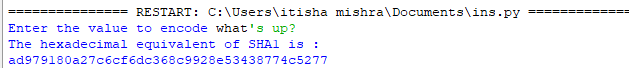
str = input("Enter the value to encode ")

result = hashlib.sha1(str.encode())

print("The hexadecimal equivalent of SHA1 is : ")

print(result.hexdigest())

**Output:**



**PRACTICAL 4**

**AIM: Digital Signatures**  
**Implement digital signature algorithms such as RSA-based signatures, and verify the integrity and authenticity of digitally signed messages.**

**Program :**

from Crypto.PublicKey import RSA

from Crypto.Signature import pkcs1\_15

from Crypto.Hash import SHA256

key = RSA.generate(2048)

private\_key = key.export\_key()

public\_key = key.publickey().export\_key()

original\_document = b"This is the original document content."

modified\_document = b"This is the modified document content."

original\_hash = SHA256.new(original\_document)

modified\_hash = SHA256.new(original\_document) #If put "modified\_document" it will be invalid

signature = pkcs1\_15.new(RSA.import\_key(private\_key)).sign(original\_hash)

try:

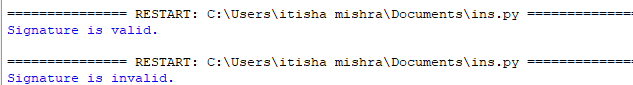
pkcs1\_15.new(RSA.import\_key(public\_key)).verify(modified\_hash, signature)

print("Signature is valid.")

except (ValueError, TypeError):

print("Signature is invalid.")

**Output:**



**PRACTICAL 5**

**AIM: Key exchange using Diffie-Hellman: Implement the Diffie-Hellman key exchange algorithm to securely exchange keys between two entities over an insecure network.**

**Program :**

from random import randint

if \_\_name\_\_=='\_\_main\_\_':

q = 23

alpha = 9

print('The Value of q is :%d'%(q))

print('The Value of alpha is :%d'%(alpha))

XA = 4

print('Secret Number for Alice is :%d'%(XA))

YA = int(pow(alpha,XA,q))

XB = 6

print('Secret Number for Bob is :%d'%(XB))

YB = int(pow(alpha,XB,q))

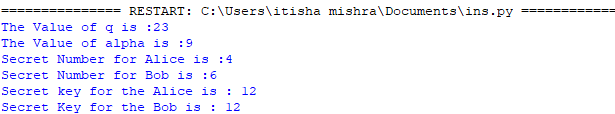
k1 = int(pow(YB,XA,q))

k2 = int(pow(YA,XB,q))

print('Secret key for the Alice is : %d'%(k1))

print('Secret Key for the Bob is : %d'%(k2))

**Output:**



**PRACTICAL 6**

**AIM: IP Security (IPsec) Configuration: Configure IPsec on network devices to provide  
secure communication and protect against unauthorized access and attacks.**

**Requirements:**

⮚ Take 3 Routers (1941)

⮚ Take 2 Switches (2960)

⮚ Take 2 PC

⮚ Configuration:

PC 0 – 192.168.1.2

PC 1 – 192.168.2.2

Router 1 (G 0/0) - 20.0.0.1

Router 1 (G 0/1) - 192.168.1.1

Router 0 (G 0/0) - 30.0.0.2

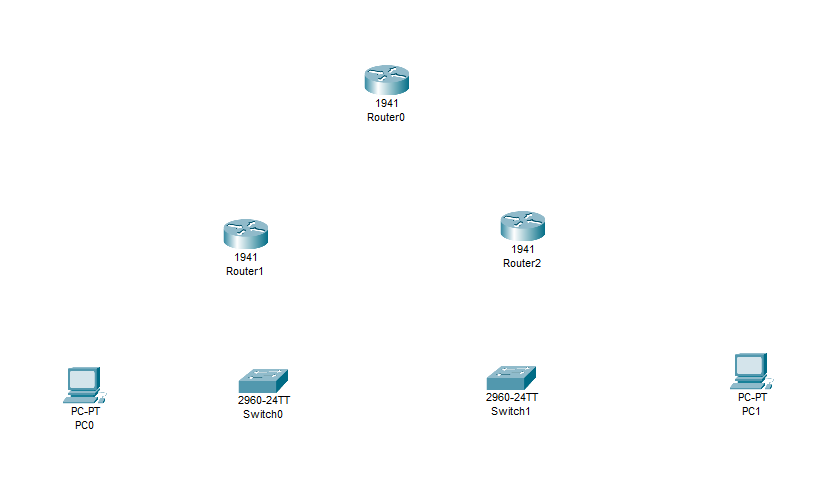
Router 0 (G 0/1) - 20.0.0.2

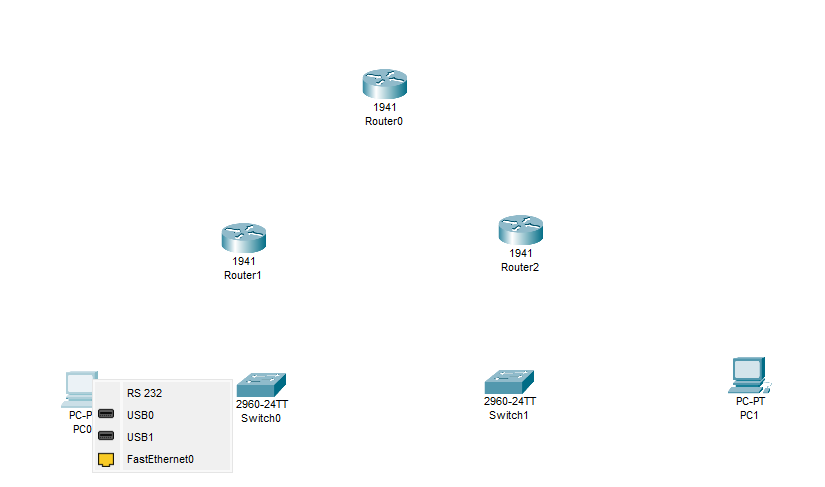
Router 2 (G 0/0) - 30.0.0.1

Router 2 (G 0/1) - 192.168.2.1

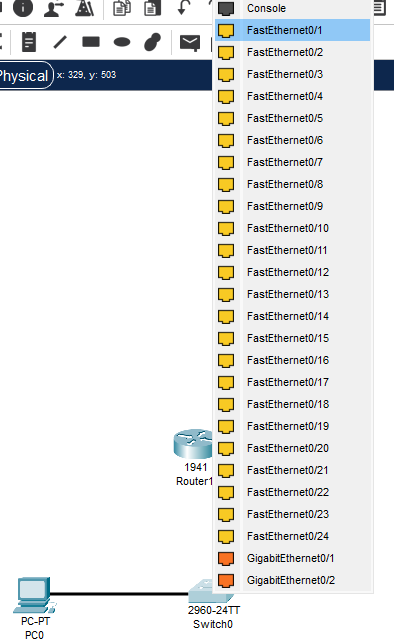
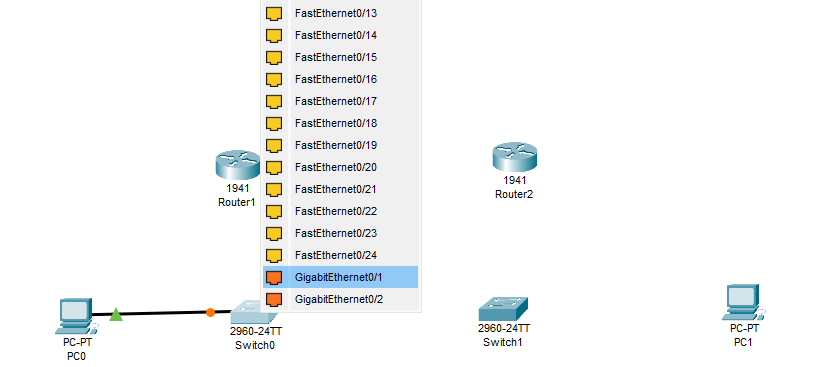
**Step 1: Implementing the Topology using Cisco Packet Tracer, configure the IP address and set the IP route.**

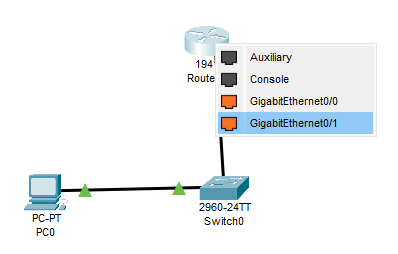
Connect the PC with Switches and also connect switches with Router using copper straight wire as given below.



Connect with copper straight wire  


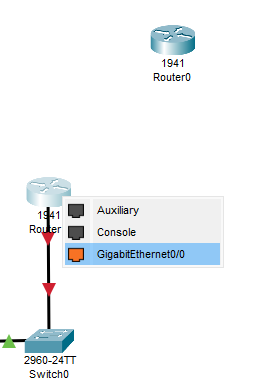
Connect the PC to the ethernet (FastEthernet0/24) and connect it with switch with GigabitEthernet0/1

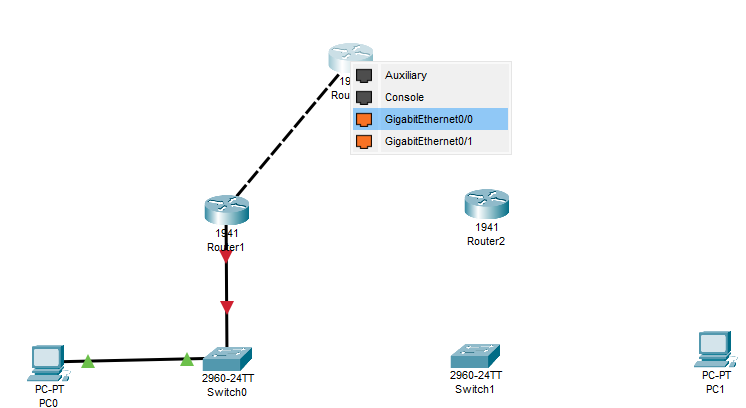
 



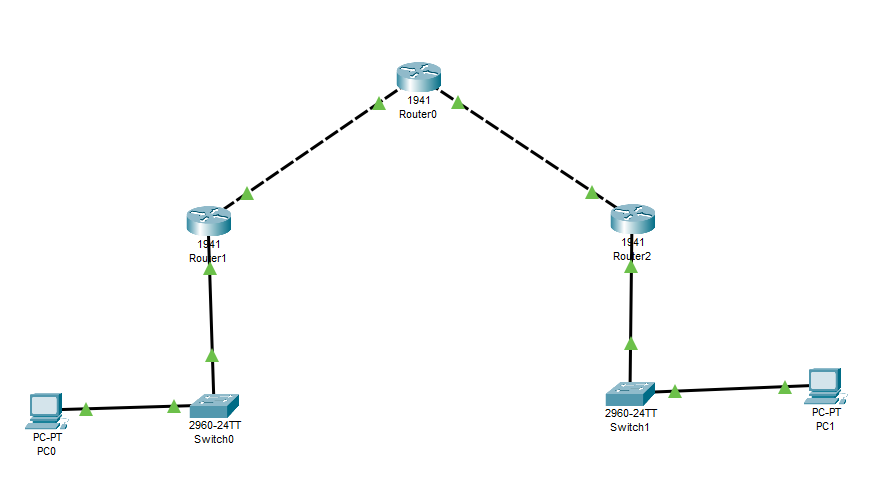
Connect the Router 1 with Router 0 and Router 0 with Router 2 using copper cross

wire as given below.



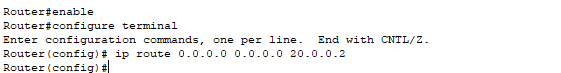


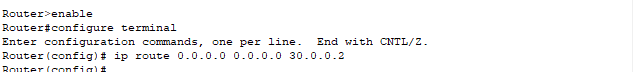
**Valid Connection:**



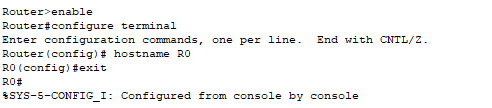
**Step 2: CLI Commands-**

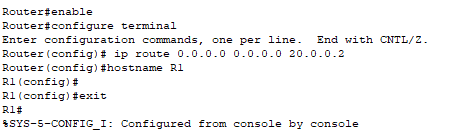
**Router1-**

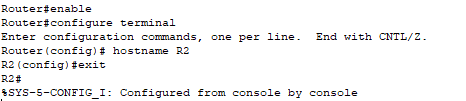


**Router2-**

**Providing Hostname:**

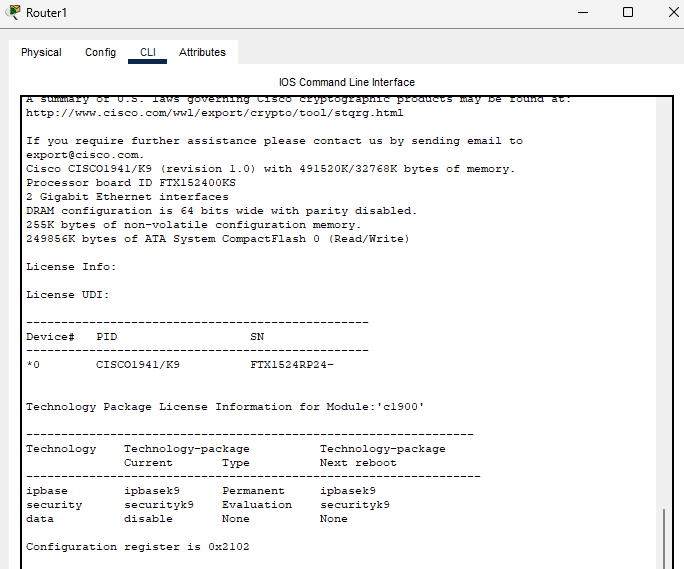
**Router0-**

**Router1-**

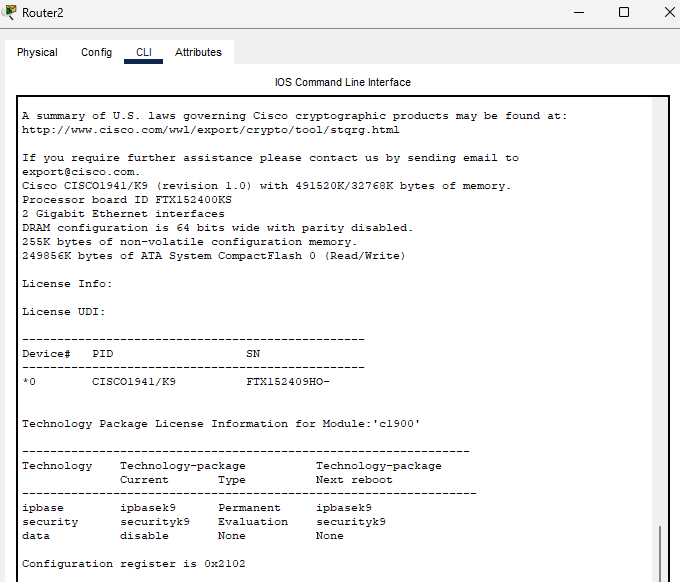
**Router2**

**Step 3: Security package:  
Enable security package for Router1 and Router2**

**Router1-**

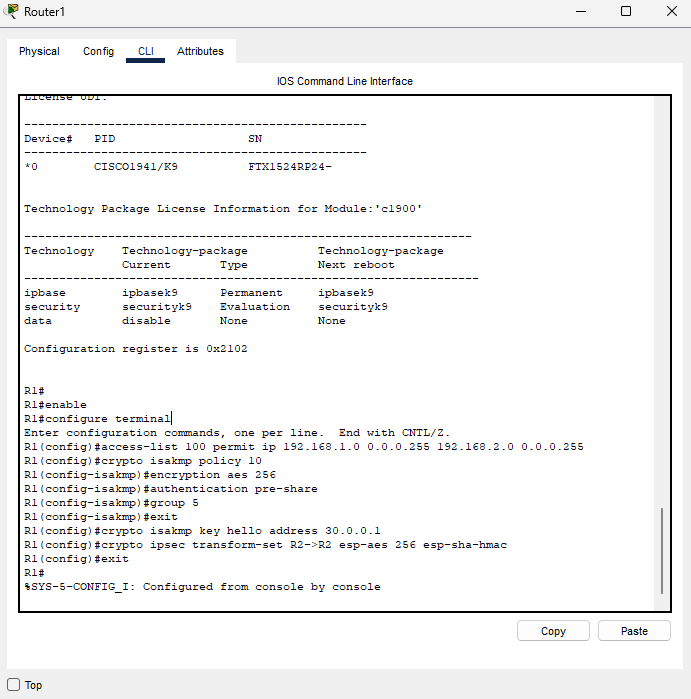


**Router2-**

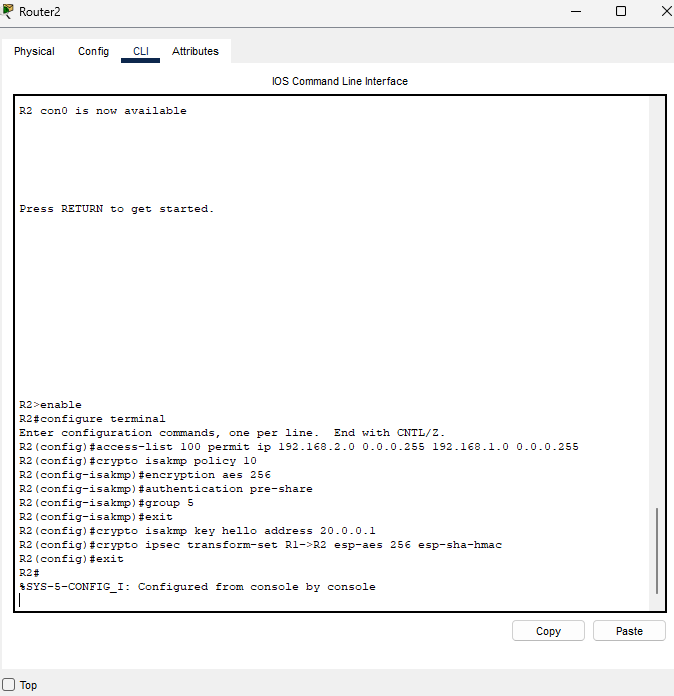


**Step 4: Apply the Access Control List (ACL)-**

**Router1-**

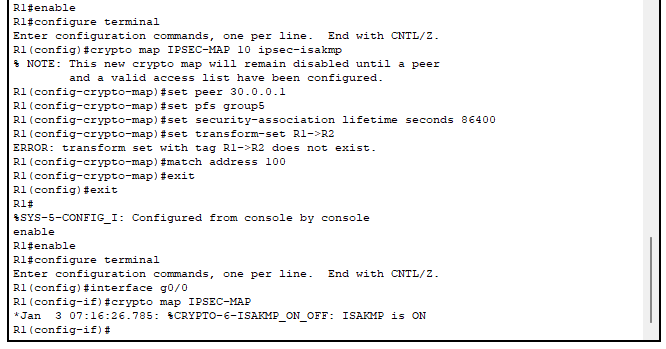


**Router2-**

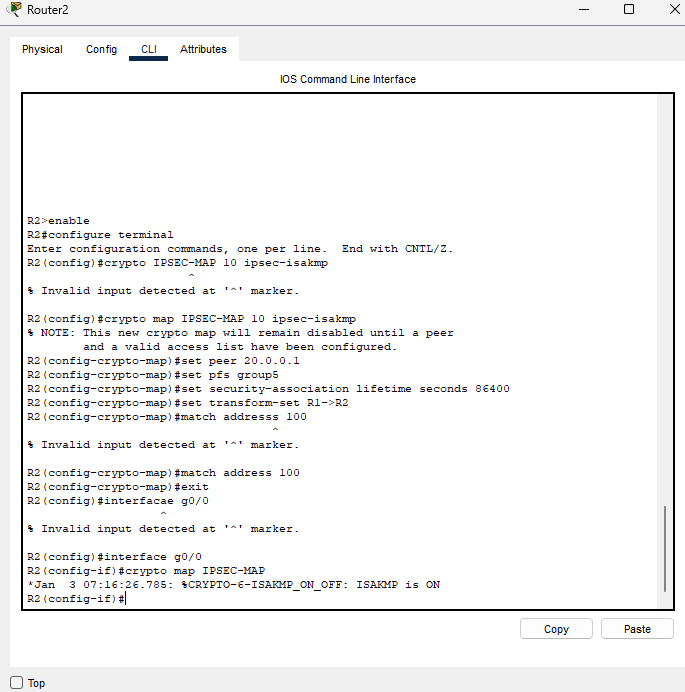


**Step 5: Create a Crypto Map-**

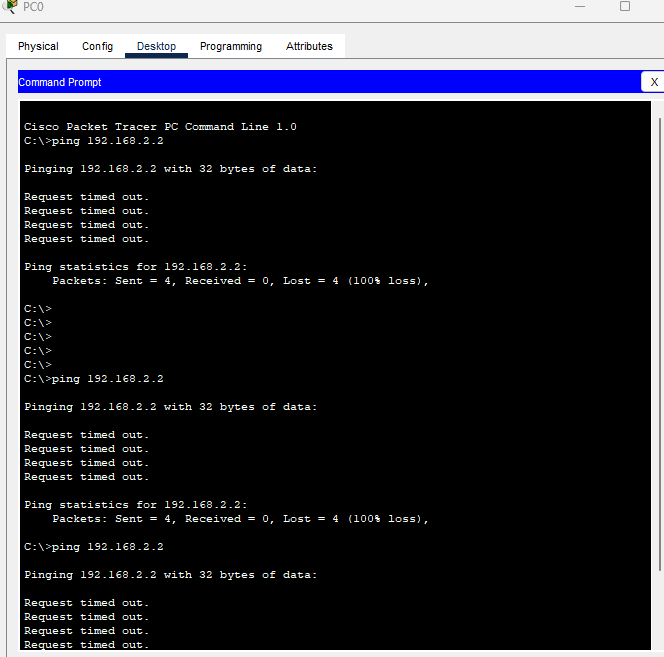
**Router1-**



**Router2-**



**Step 6: Verify the output by pinging the PC-**

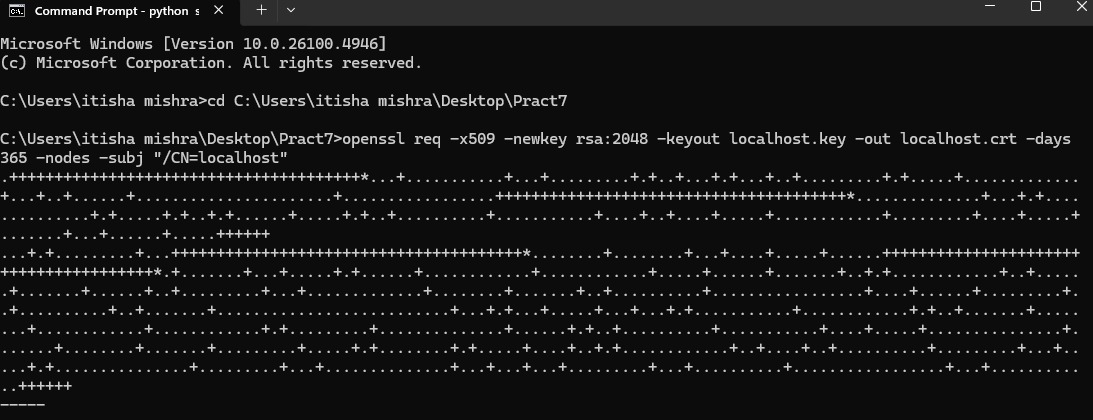


**PRACTICAL 7**

**AIM:** **Web Security with SSL/TLS -**

**Configure and implement secure web communication using SSL/TLS protocols,  
including certificate management and secure session establishment.**

Step 1: Generate a new self-signed SSL certificate and key for localhost:





Step 2:Server Side Program-

import socket

import ssl

context = ssl.SSLContext(ssl.PROTOCOL\_TLS\_SERVER)

context.load\_cert\_chain(certfile="localhost.crt", keyfile="localhost.key")

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as server:

    server.bind(("", 8443))

    server.listen(5)

    print("Server ready and listening for connections")

    # Wait for new connections in a loop

    while True:

        sock, address = server.accept()

        print("New connection from", f"{address[0]}:{address[1]}")

        # Wrap socket with ssl

        ssl\_sock = context.wrap\_socket(sock, server\_side=True)

        while True:

            data = ssl\_sock.recv(1024)

            # Decode byte array to utf-8 string

            decoded = data.decode('utf-8')

            # Close the socket if the sock sends empty bytes

            if decoded == "":

                break

            # Log what the sock sends

            print(f"[{address[0]}:{address[1]}] {decoded}")

            # Echo the data back to the sock

            ssl\_sock.sendall(data)

        # Gracefully close the connection and wait for next one

        print("Closing connection with", f"{address[0]}:{address[1]}")

        ssl\_sock.close()

Step 3: Client Side Program

import socket

import ssl

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as sock:

    sock.settimeout(10)

    # Wrap socket with ssl

    context = ssl.SSLContext(ssl.PROTOCOL\_TLS\_CLIENT)

    context.load\_verify\_locations('localhost.crt')

    ssl\_sock = context.wrap\_socket(sock, server\_hostname="localhost")

    # Connect to the server

    ssl\_sock.connect(("localhost", 8443))

    print("Connected to server")

    # Send input data to server and wait for response in a loop

    while True:

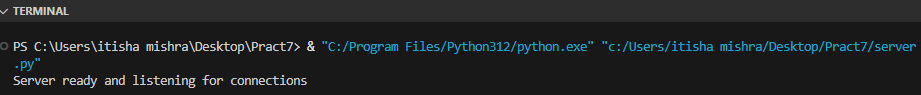
        ssl\_sock.sendall(bytes(input(">"), "utf-8"))

        data = ssl\_sock.recv(1024)

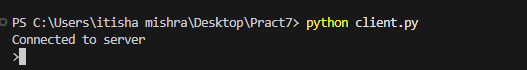
        print("Server responded:", data.decode('utf-8'))

**Output:**

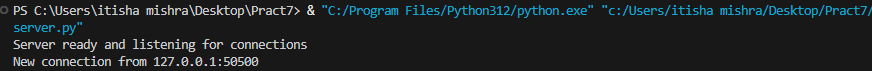
* **Run the server.py file -**



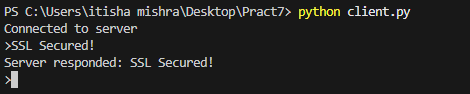
* **Keep the server running, and run the client.py file –**



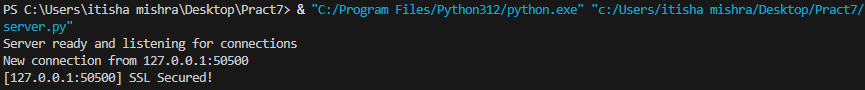
* **Server output -**



* **Test the connection by sending a message from the client by typing in the console.**



* **The server will echo the same content of the message back to the client -**

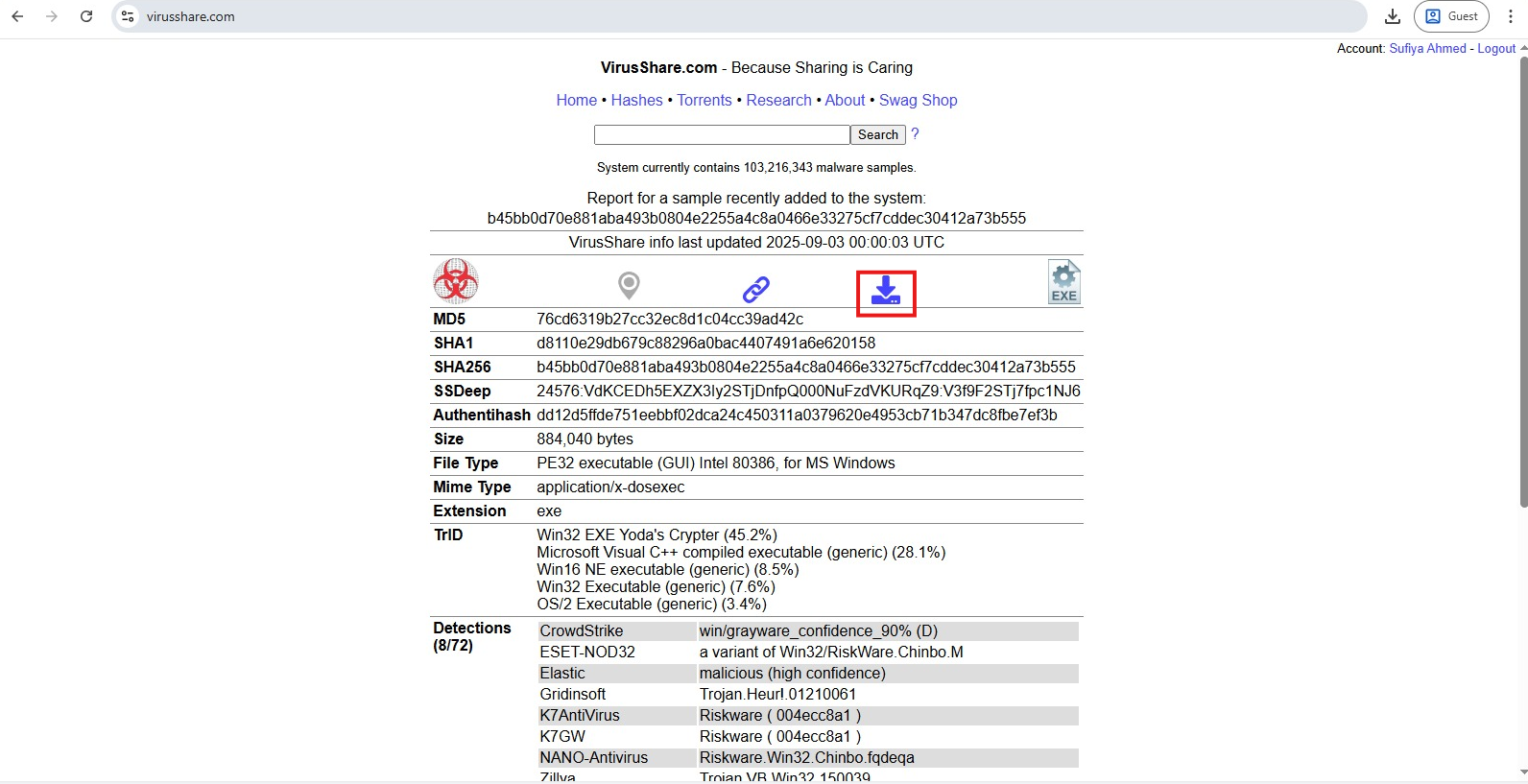


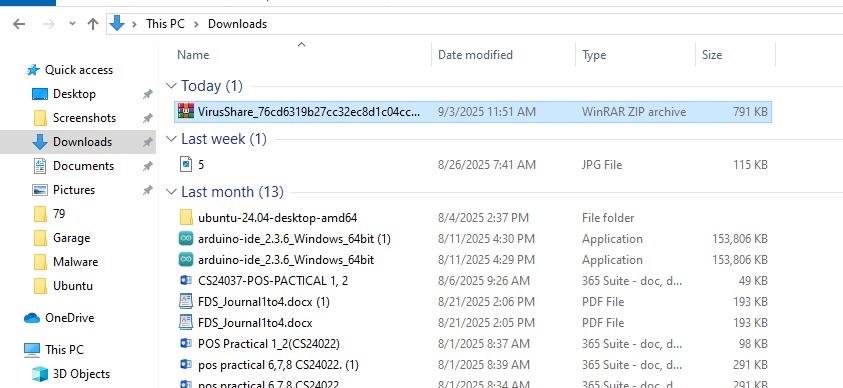
**PRACTICAL 8**

**AIM: Malware Analysis and Detection: Analyze and identify malware samples using antivirus tools, analyze their behavior, and develop countermeasures to mitigate their impact.**

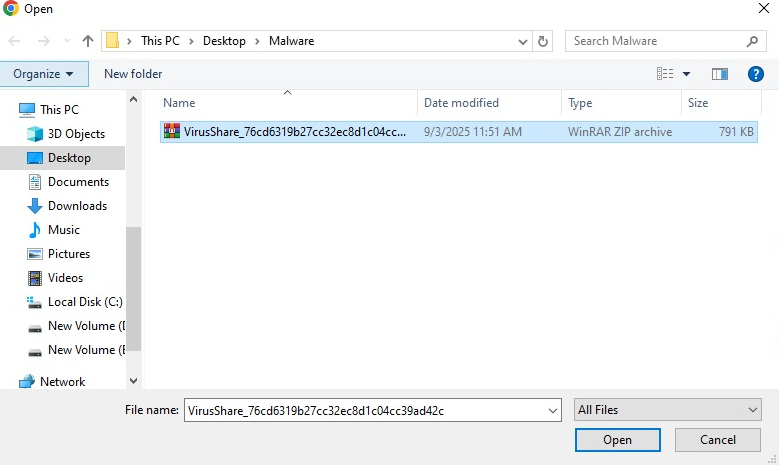
**Step 1**: Visit the website VirusShare.com for downloading the virus samples.

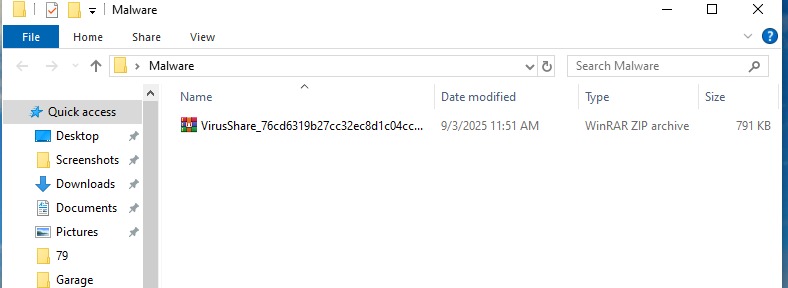
**Step 2**: Login:  
Username: Sufiya111  
Password: INSPrac9

**Step 3**: Click on download.  


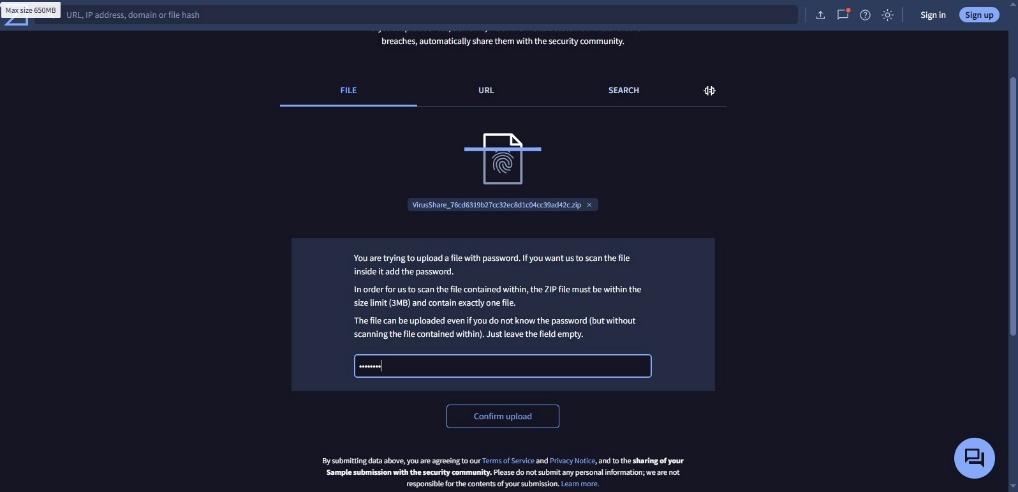


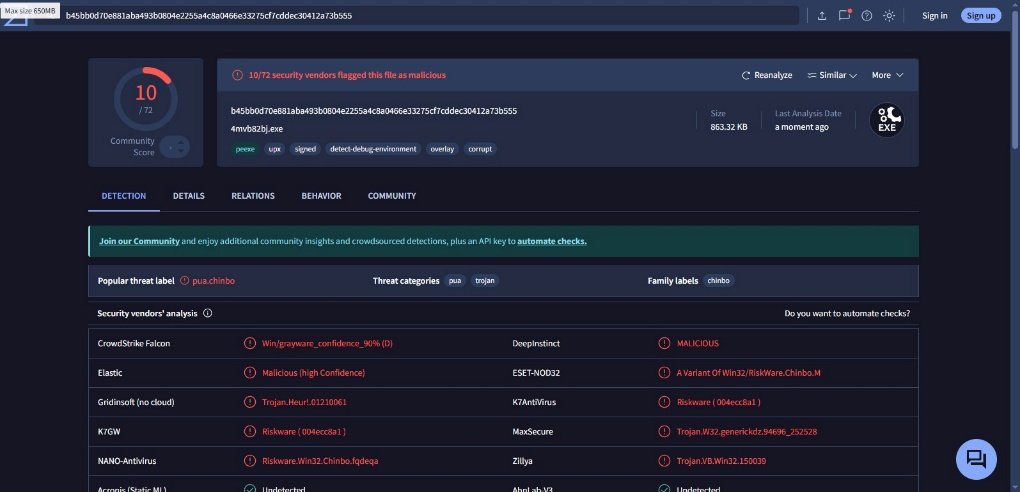
**Step 4**: Create a folder on the desktop name as Malware.





**Step 5**: Now scan the zip file on the website www.virustotal.com. Choose a file and scan it.  

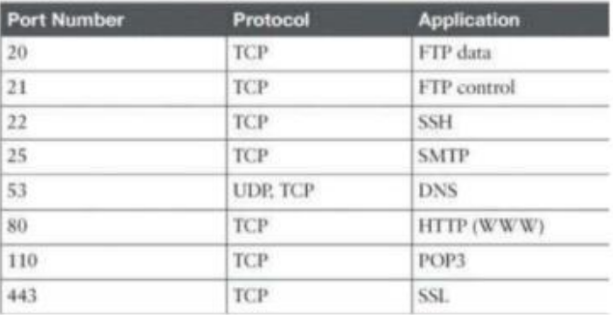

  
Password – infected



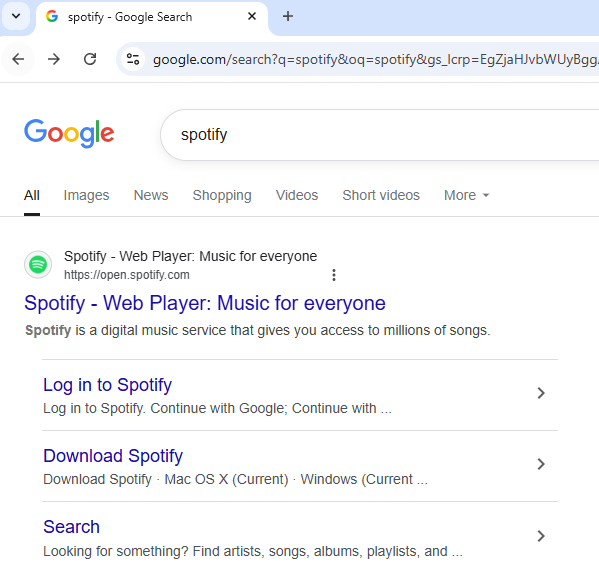
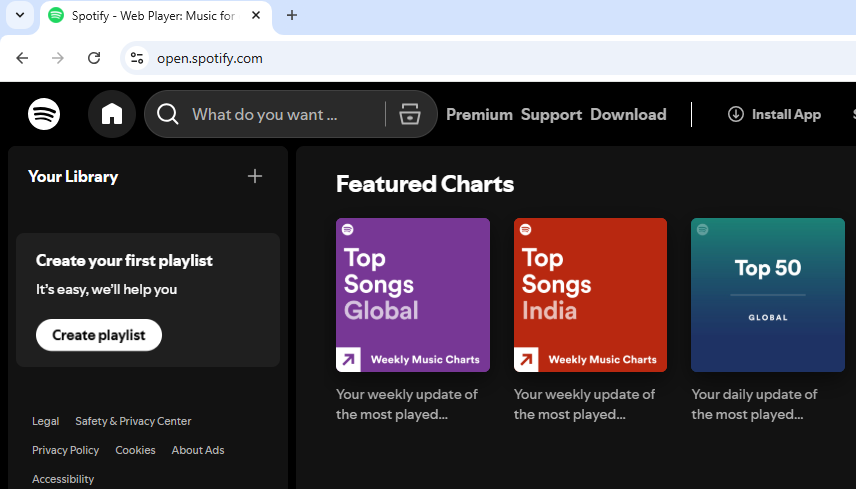
**PRACTICAL 9**

**AIM**:**Firewall Configuration and Rule-based Filtering: Configure and test firewall rules to control network traffic, filter packets based on specified criteria, and protect network resources from unauthorized access.**

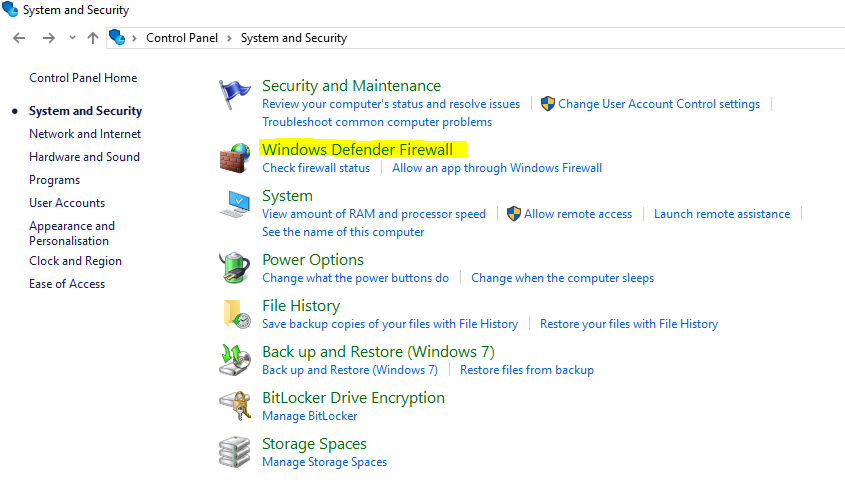
We would use firewall to block    
1) A Port    
2) A Program    
3) A Website

**Part 1**: Blocking the HTTP and HTTPS (Port 80 and Port 443) using the Firewall. Before starting with the blocking port process, we note that the applications running at the server end are identified with the well-known Port numbers, some of the commonly used are as follows.   


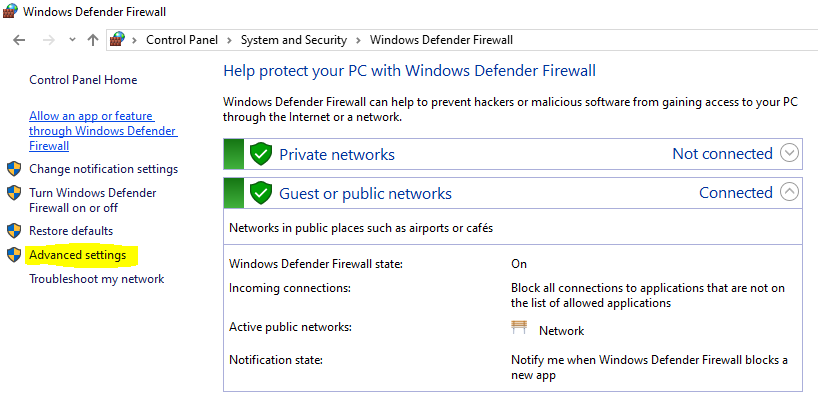
**For Inbound-**

Step 1: We access any website through the browser and confirm that the HTTP/HTTPS  protocols are working. Go to Browser > Search Anything > Open > Working.  
  
 

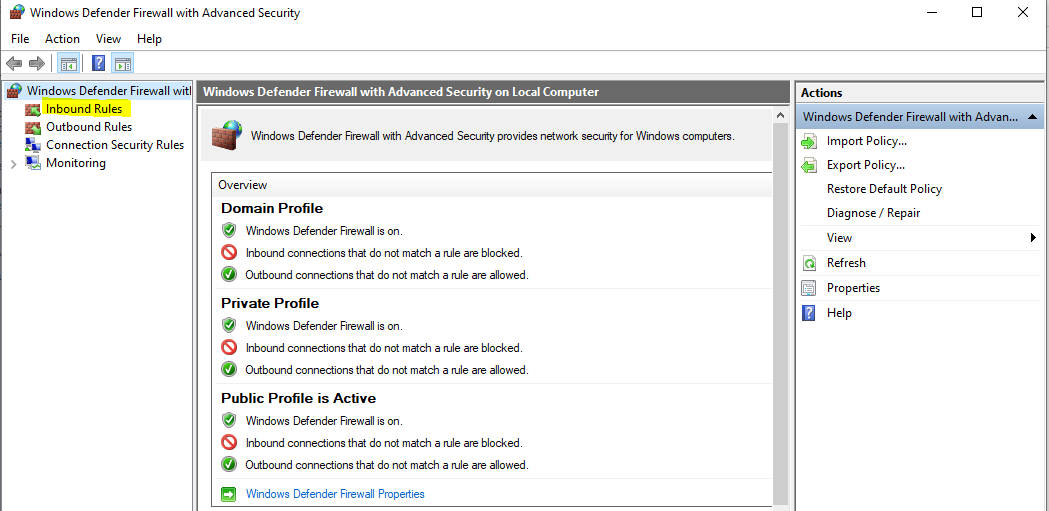
Step 2: We open ‘Windows Defender Firewall’



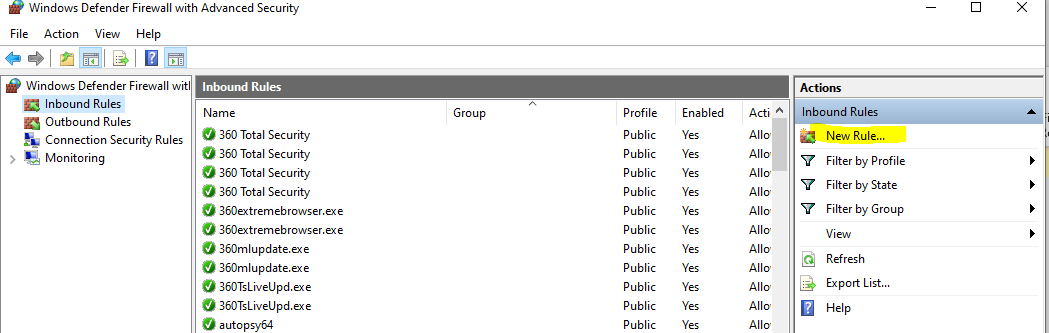
Step 3: Next, we click on ‘Advanced settings’



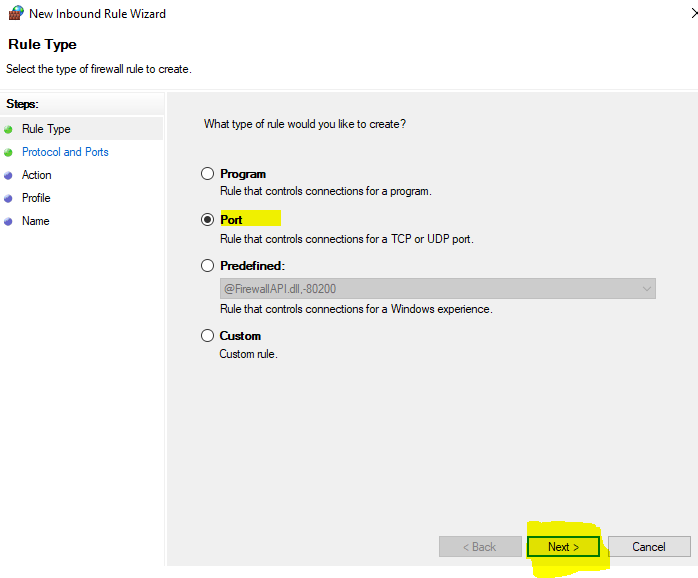
Step 4: Next go to ‘Inbound rules’



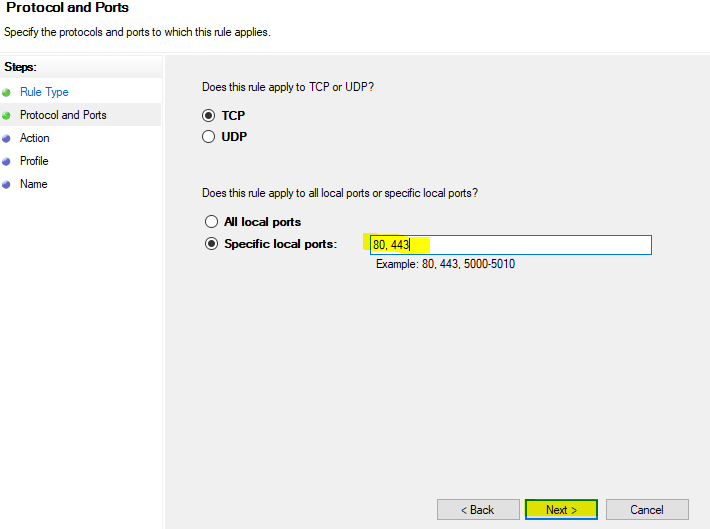
Step 5: Now create a ‘new rule’ for Inbound



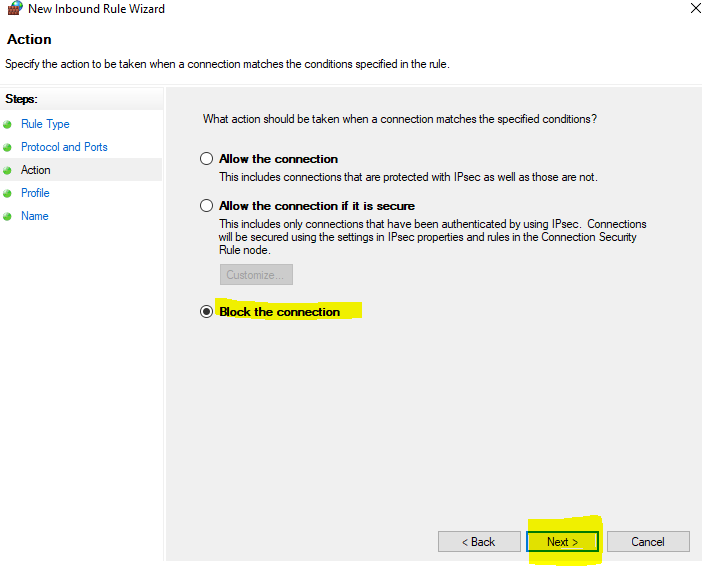
Step 6: Select ‘Port’ and click ‘Next’



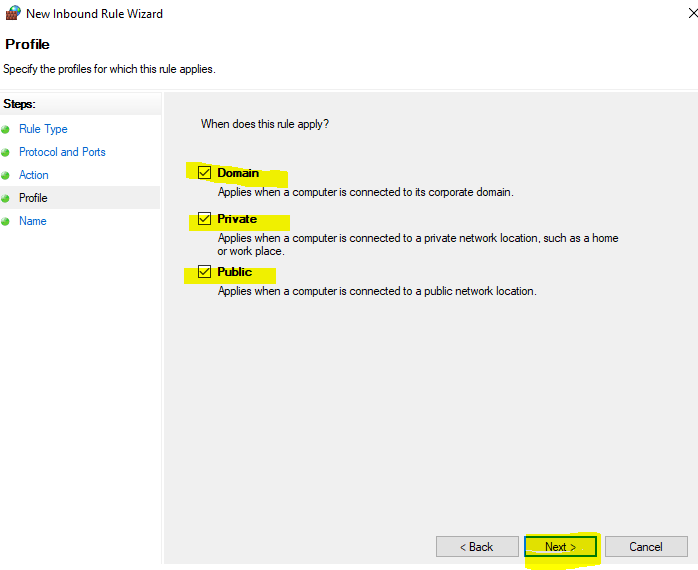
Step 7: Give Port ‘80, 443’ and click ‘Next’



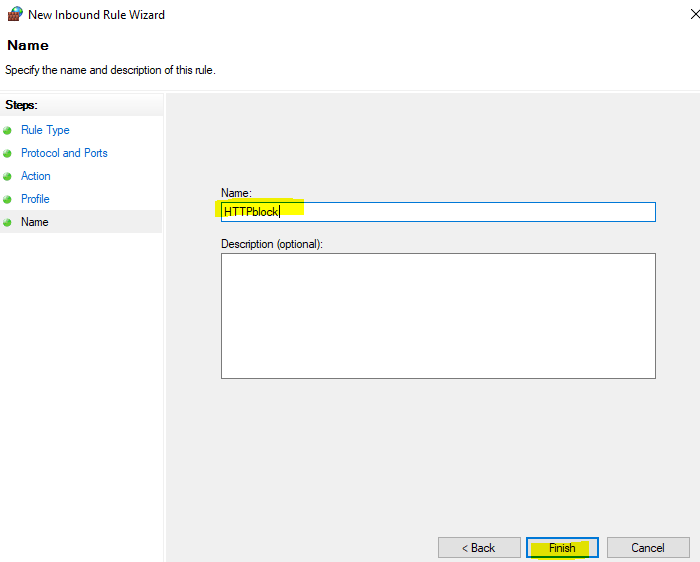
Step 8: Block the connection



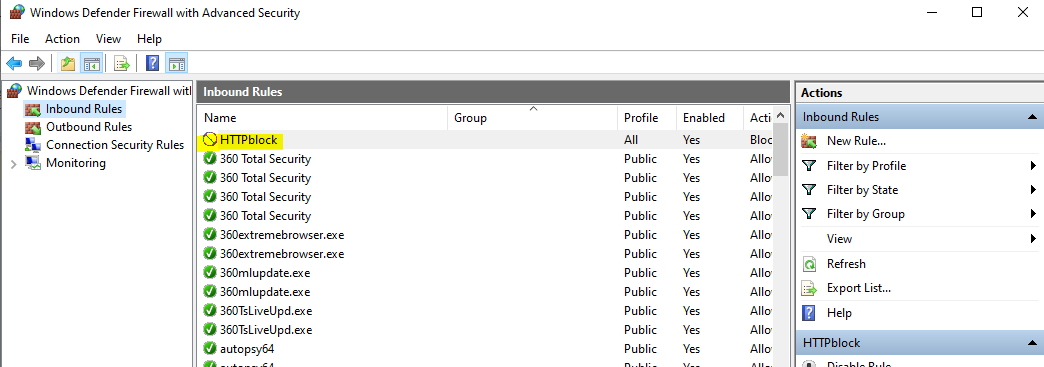
Step 9: Check all the rules application are enable



Step 10: Name the rule



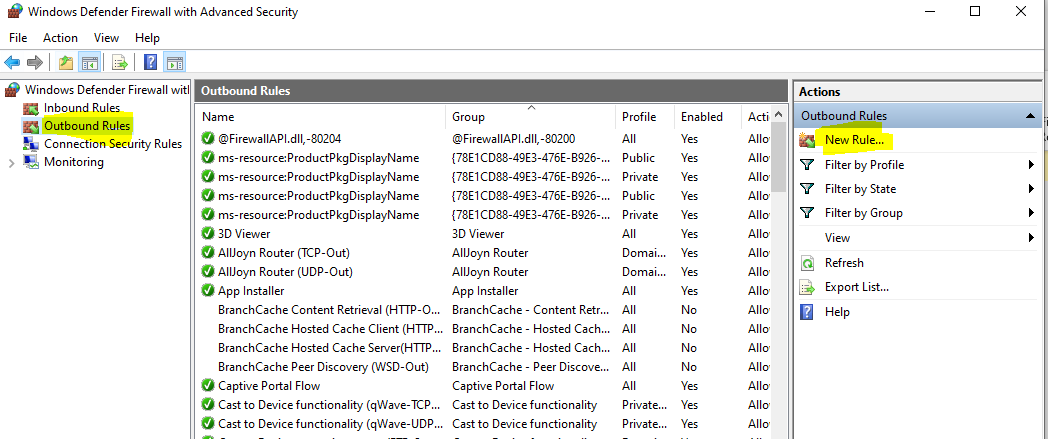
New rule for inbound is added.



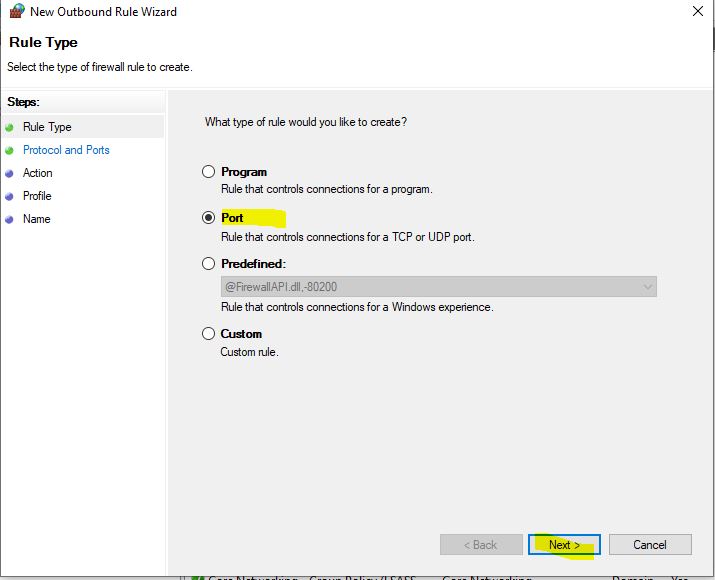
**For outbound-**

We repeat all the above steps for creating ‘Outbound Rules’, and then try to access the internet. We will see that the access is blocked

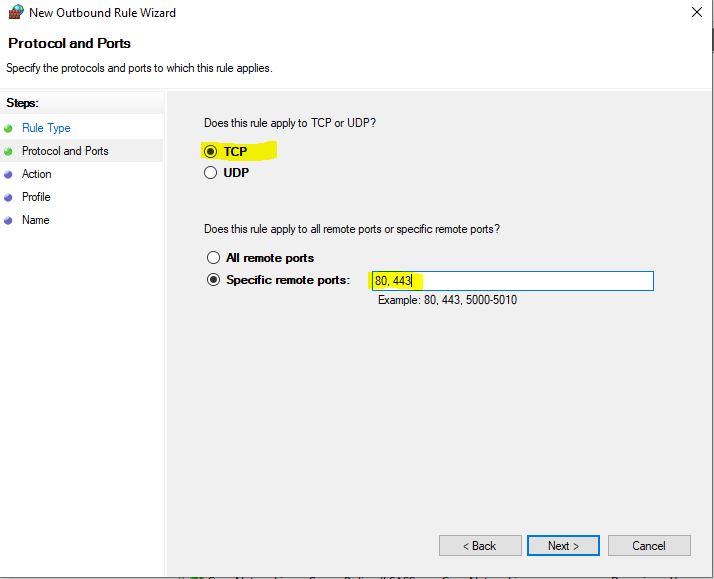
Step 1: Create a ‘new rule’ for Outbound



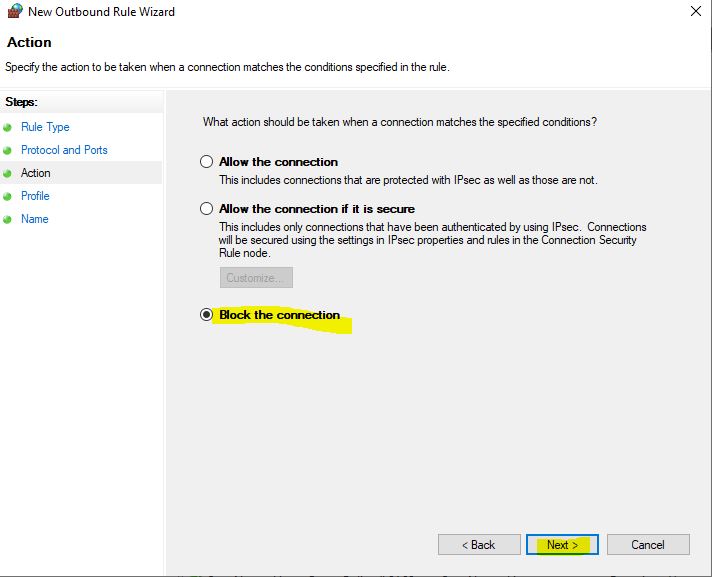
Step 2: Select ‘Port’ and click ‘Next’



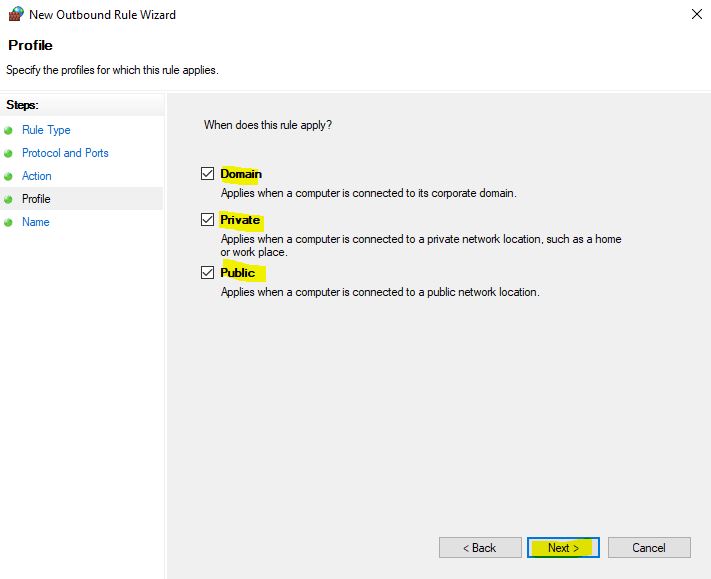
Step 3: Give Port ‘80, 443’ and click ‘Next’



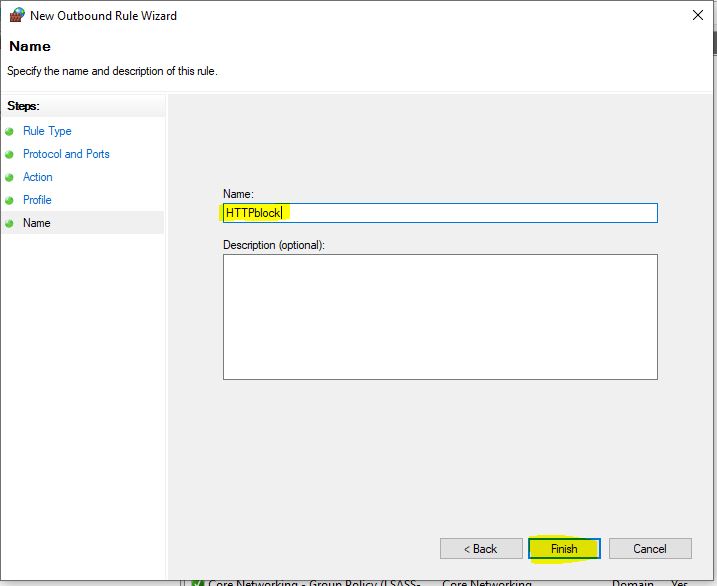
Step 4: Block the connection



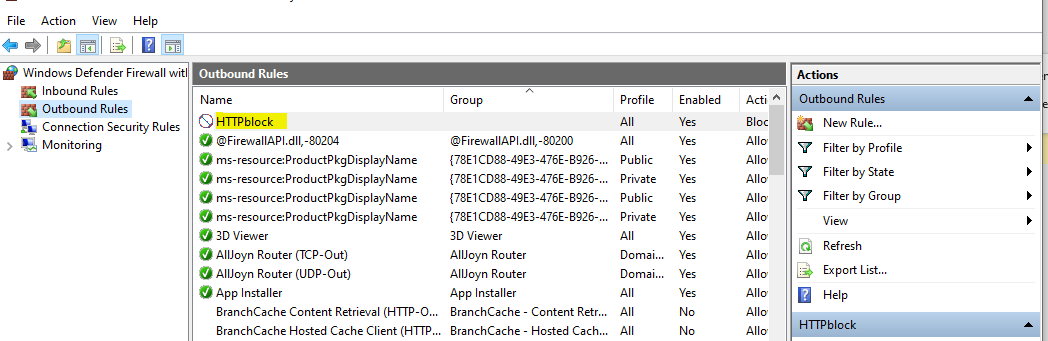
Step 5: Check all the rules application are enable



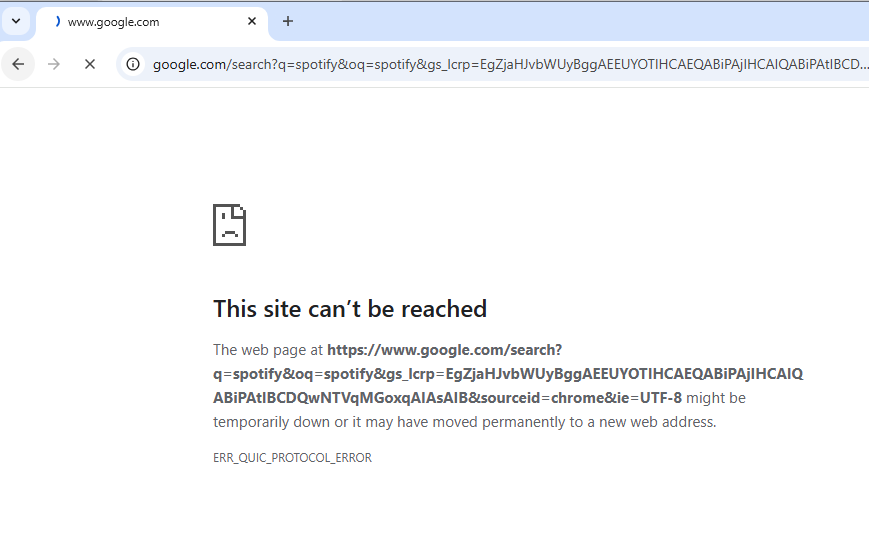
Step 6: Name the rule



New rule for outbound is added.

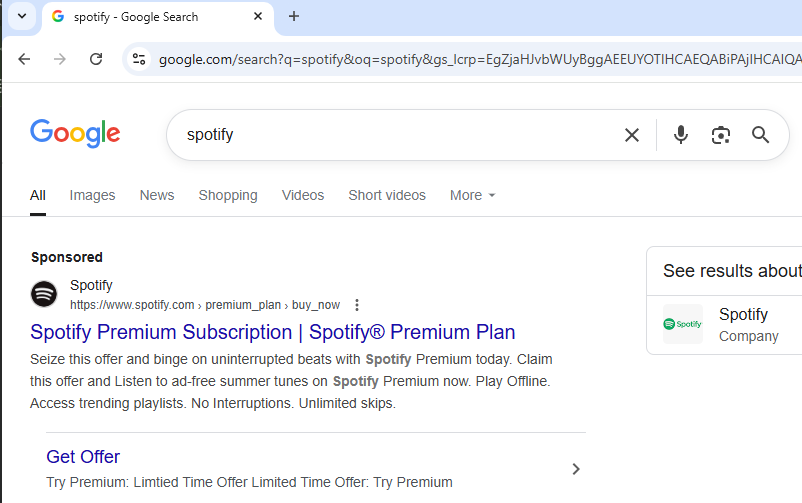


**Now the access is blocked-**



**Now to unblock the internet:  
Go to firewall > Go to inbound > Right click and delete the rule  
Go to firewall > Go to outbound > Right click and delete the rule**

**After deleting inbound and outbound rule-**



**Part 2**: Blocking the Program.

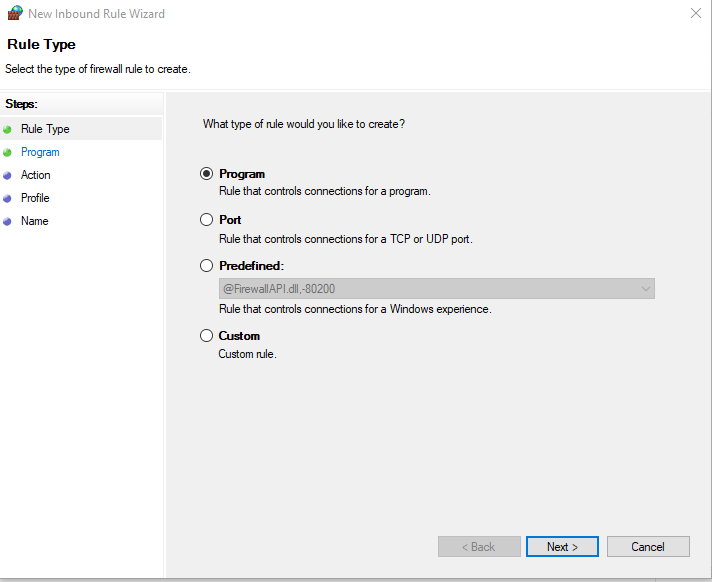
Step 1: Open Windows Defender Firewall.

Step 2: Go to Advance Settings

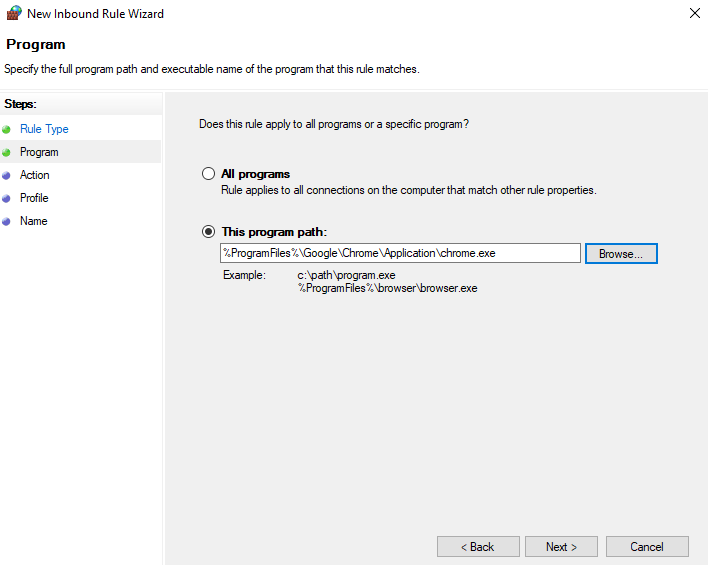
Step 3: Click on Inbound Rule

Step 4: Create new rule

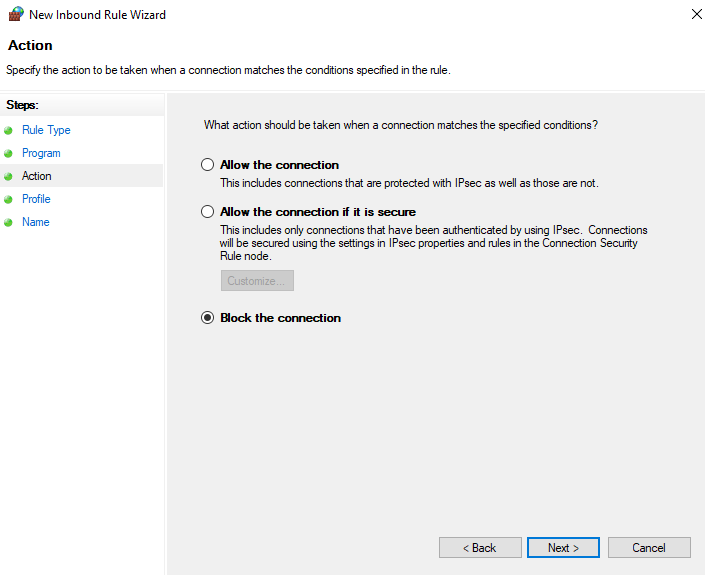
Step 5: Select Program and click on Next button



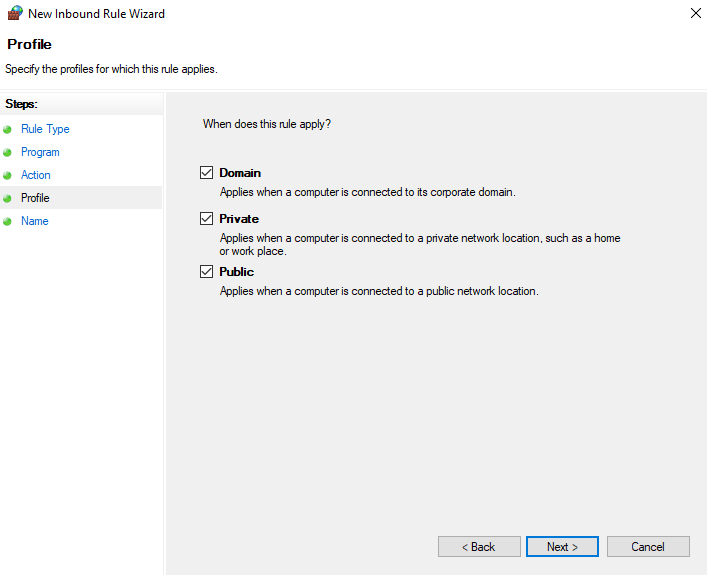
Step 6: We are blocking the chrome program so browse the path and click on next.



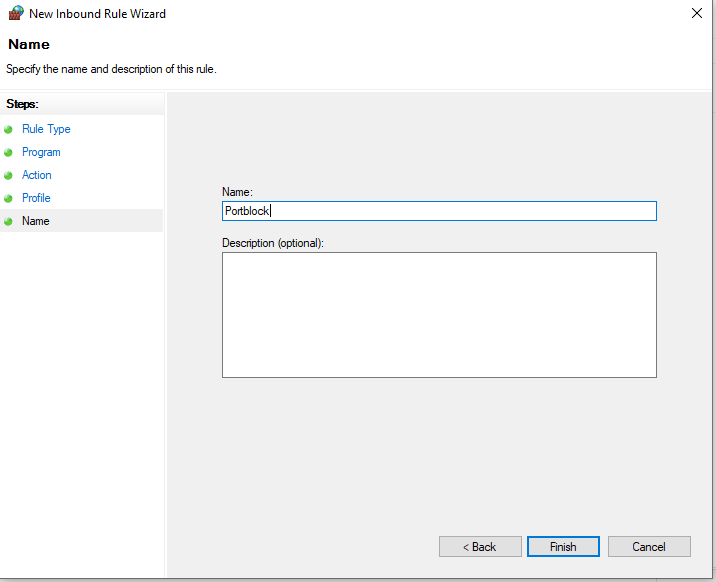
Step 7: Now click on block the connection.



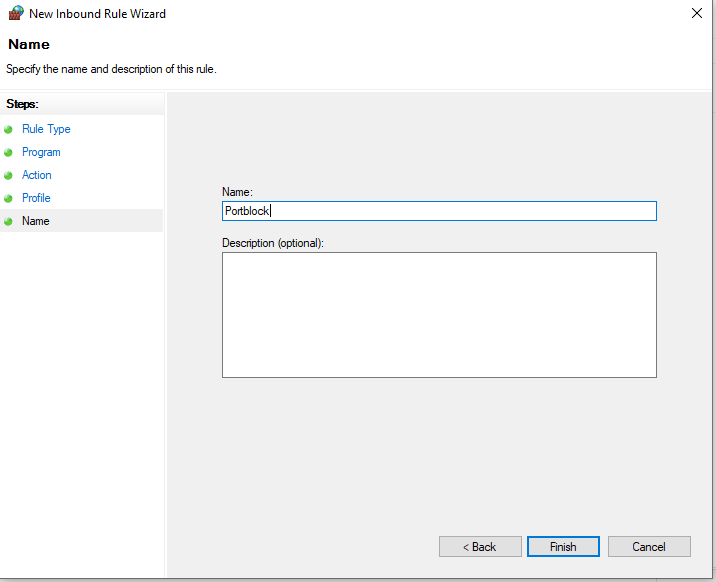
Step 8: Tick all the checkboxes.



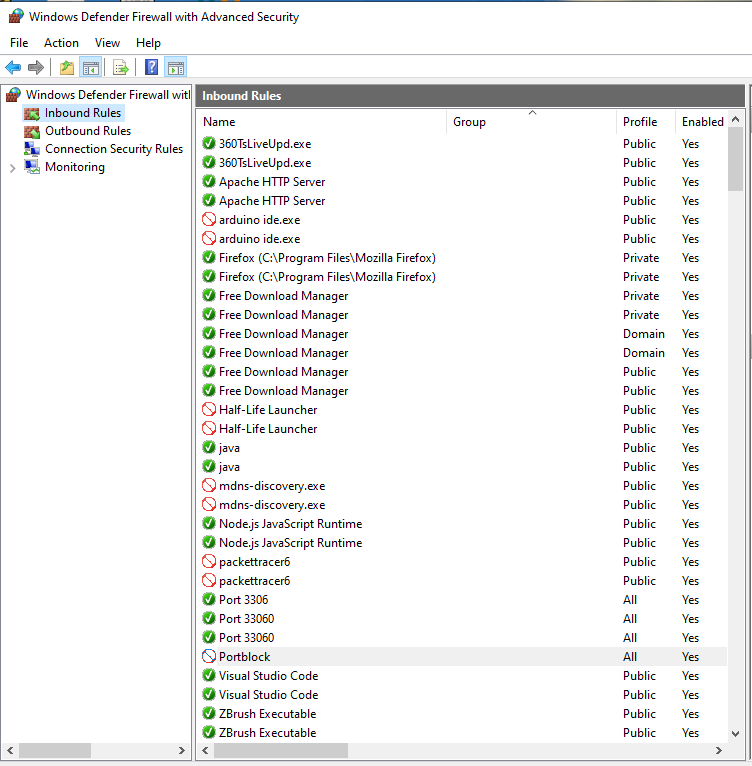
Step 9: After clicking on Next button give the name to that rule.



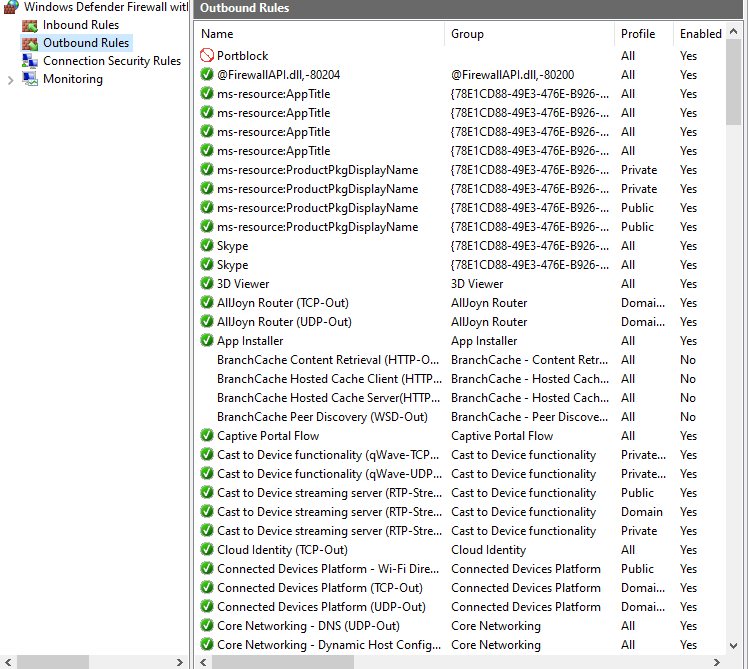
Then click on finish:



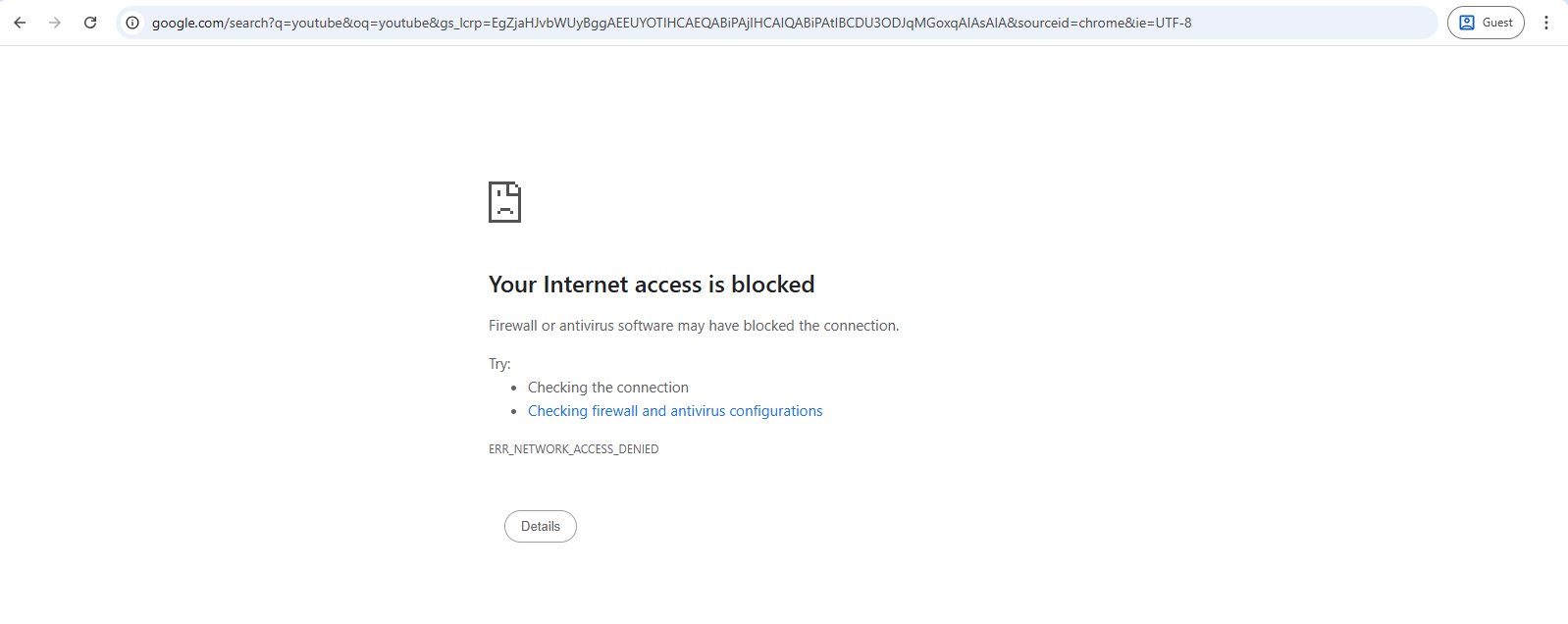
The Inbound rule is created:



Step 10: Follow the same steps for Outbound Rule.

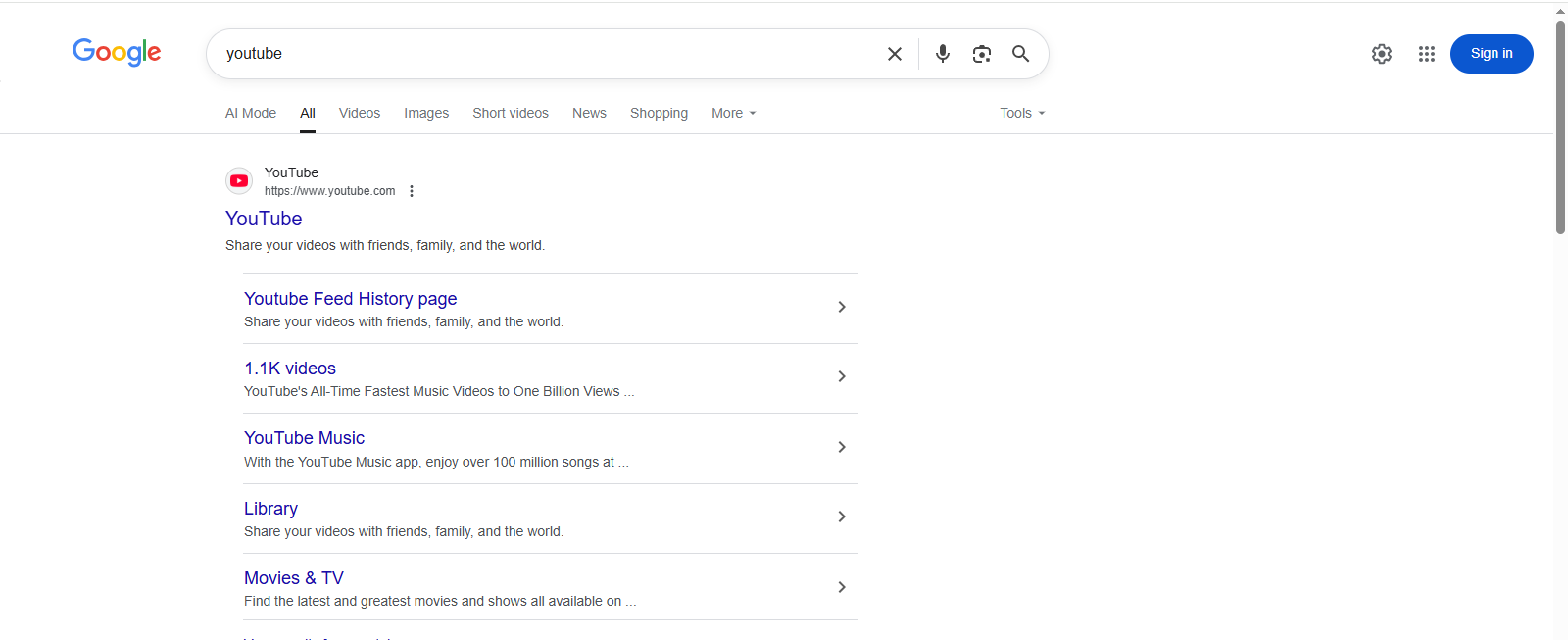


Step 11: Now open chrome and check it will show connect to internet

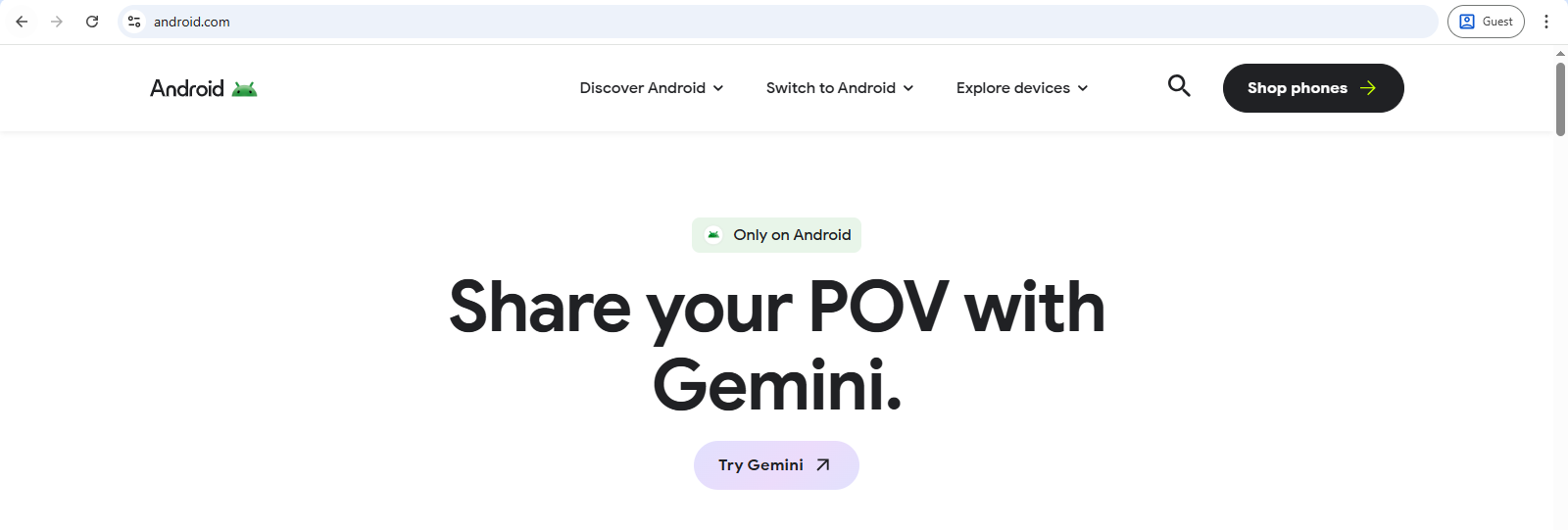
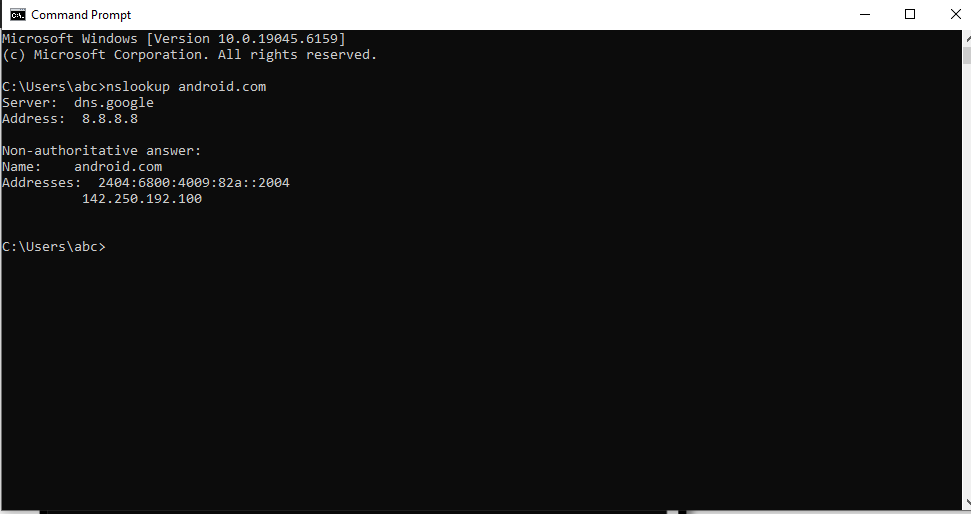


**Now to unblock the Chrome:  
Go to firewall > Go to inbound > Right click and delete the rule  
Go to firewall > Go to outbound > Right click and delete the rule**

**After deleting inbound and outbound rule-**



**Part 3:** Blocking the website android.com  
We open the browser and access the website, which is now accessible.

  
We find the IP addresses of the website using the following command  
*nslookup android.com*  


We save the IP addresses  
IPV4 - 142.250.192.100  
IPV6 - 2404:6800:4009:822::2004

We will make Inbound and Outbound rule for IP addr

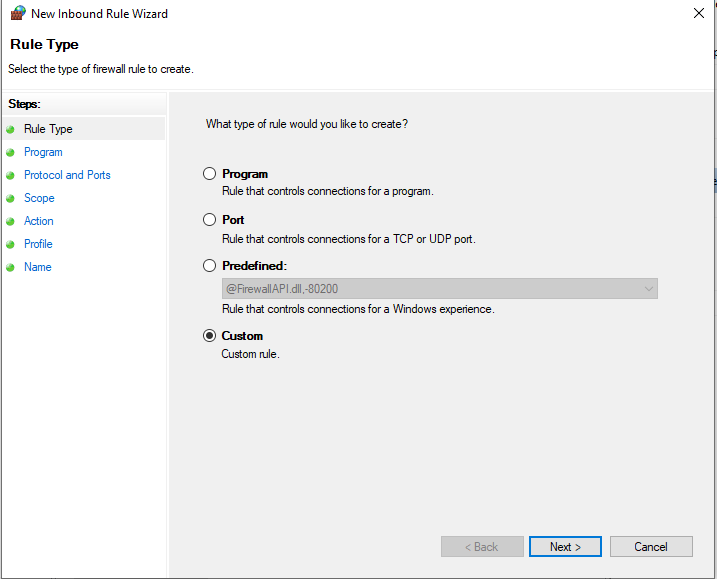
Step 1: Open Windows Defender Firewall.

Step 2: Go to Advance Settings

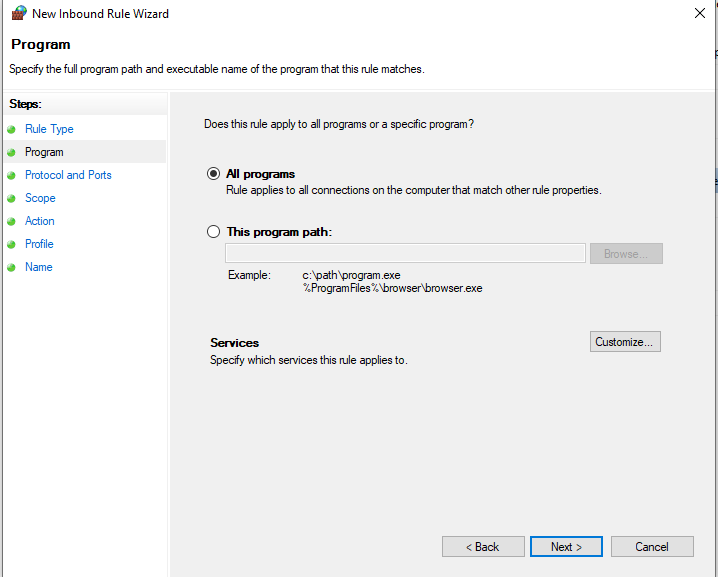
Step 3: Click on Inbound Rule

Step 4: Create new rule

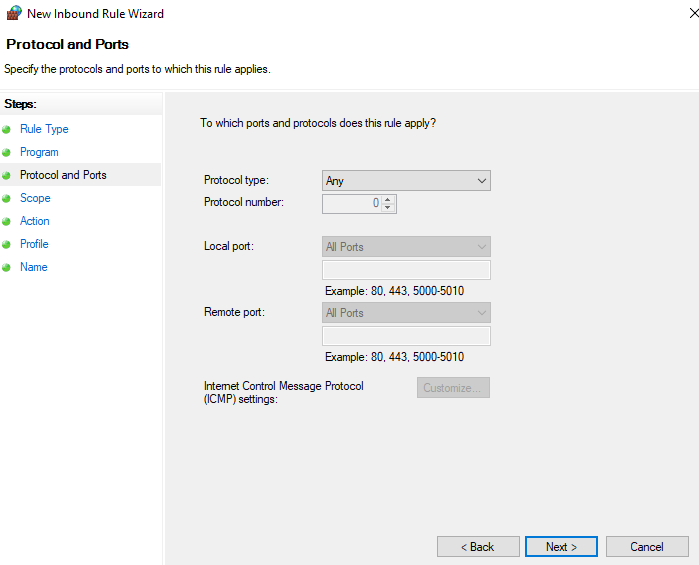
Step 5: Select Custom and click on Next button



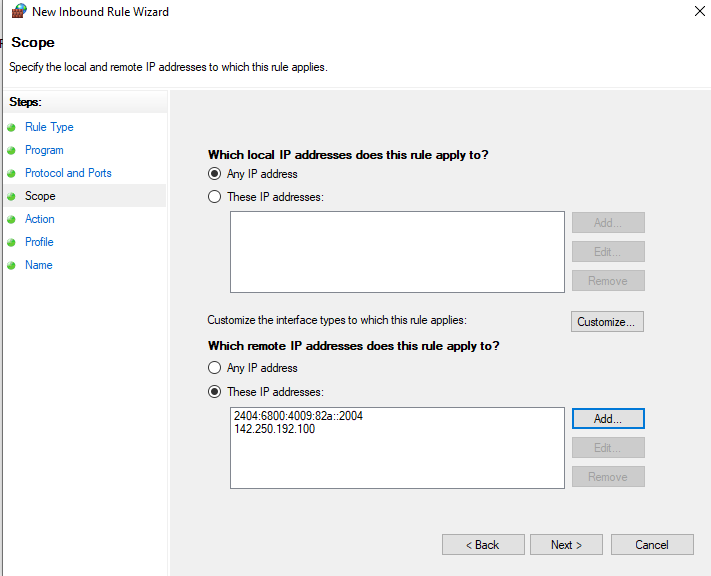
Step 6: Select “All programs” and click on Next button



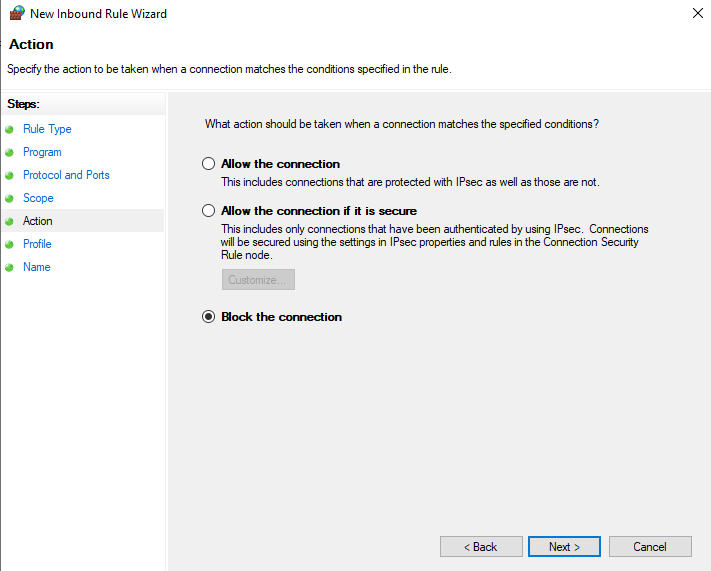
Step 7: Click on Next button



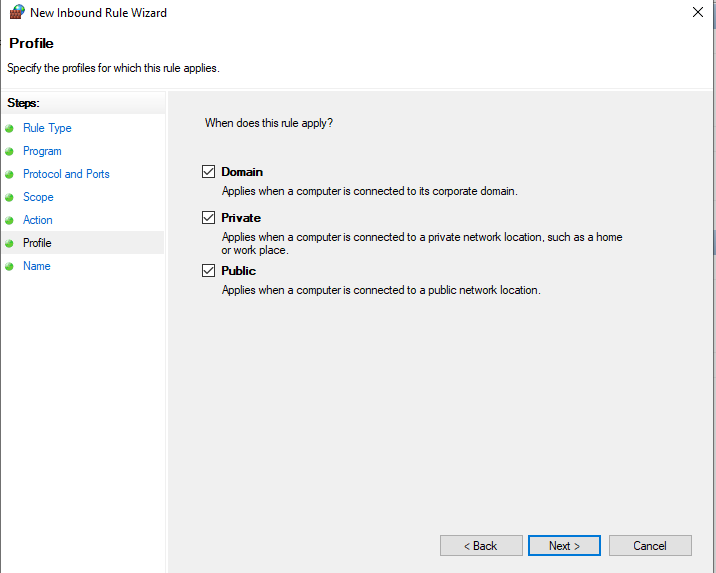
Step 8: Add both the addresses



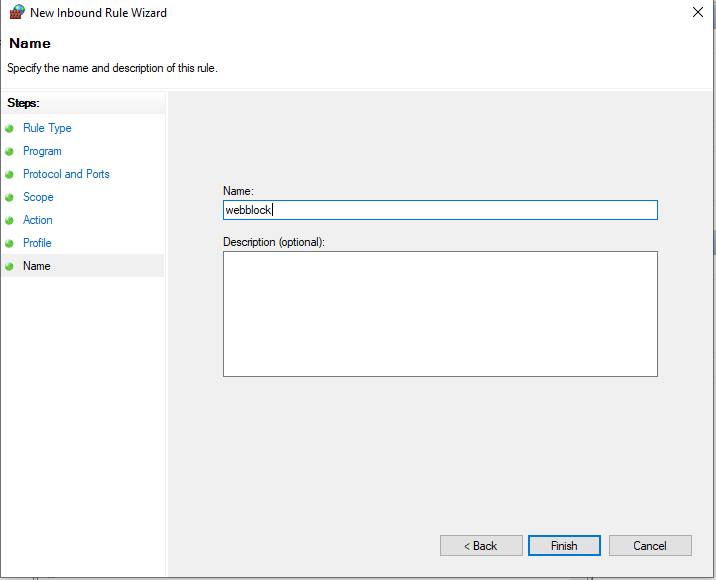
Step 9: Block the connection



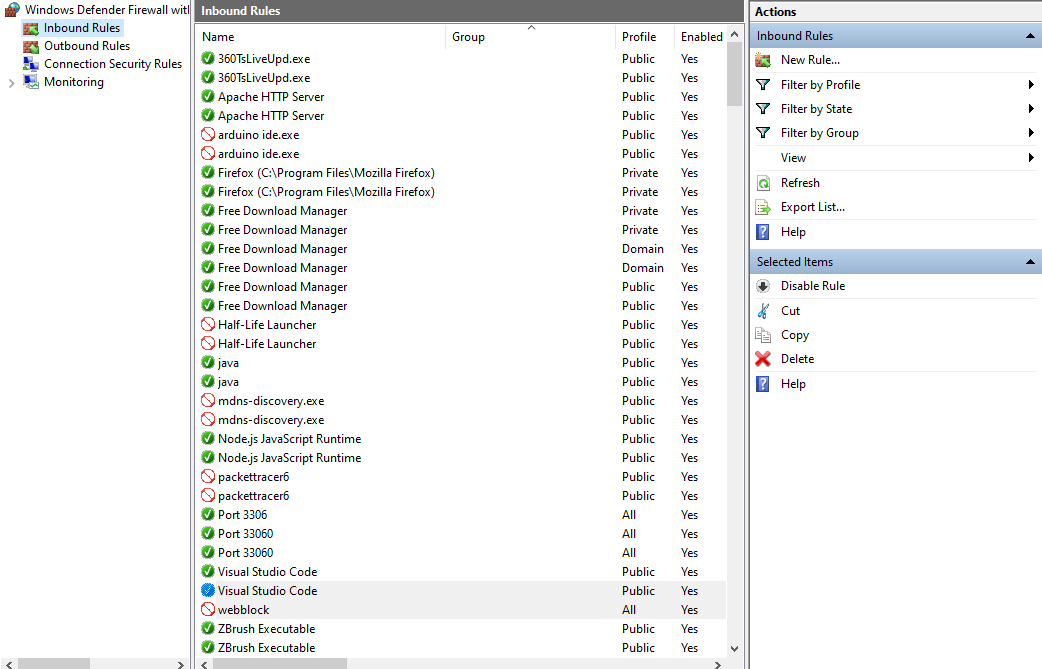
Step 10: Check all the boxes



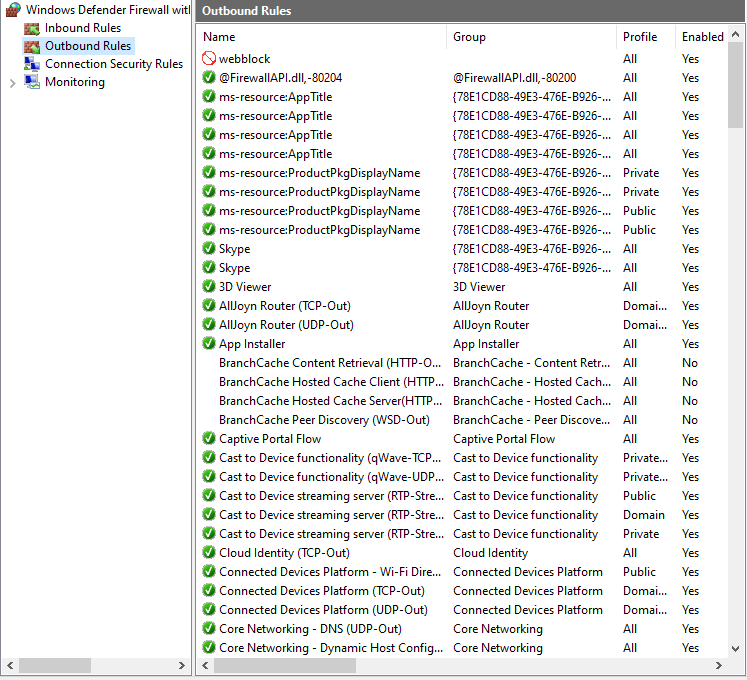
Step 11: Name the rule and click on finish.



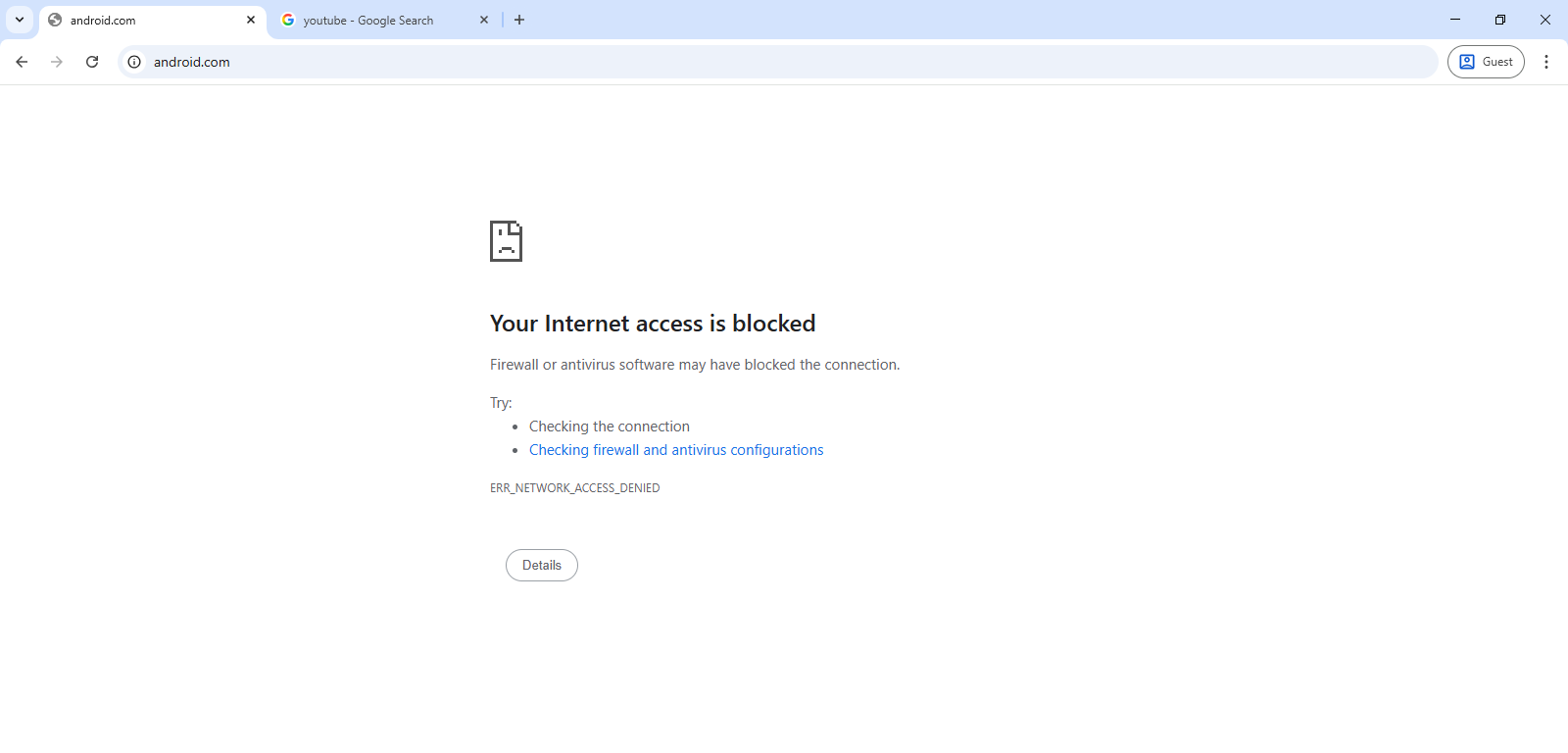
The Inbound Rule is created:



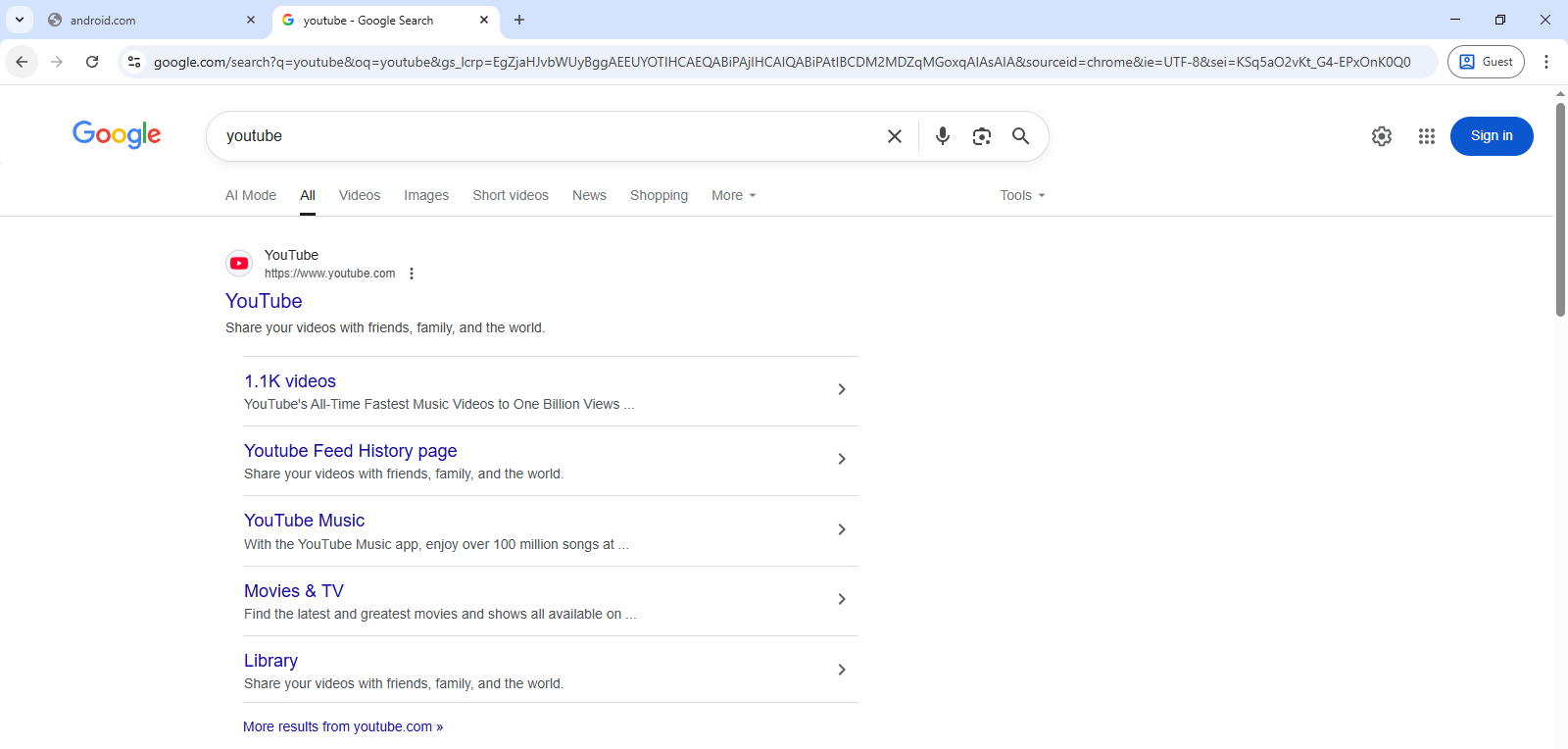
Now, follow the same steps for creating the outbound rule:



Go to browser and search android.com



Simultaneously we checked for other search its working properly.



**Now to unblock the website:  
Go to firewall > Go to inbound > Right click and delete the rule  
Go to firewall > Go to outbound > Right click and delete the rule**

**After deleting inbound and outbound rule-**

