LAKIREDDY BALI REDDY COLLEGE OF ENGINEERING

(AUTONOMOUS)
L.B. REDDY NAGAR, MYLAVARAM, KRISHNA DIST., A.P.-521 230.

DEPARTMENT
OF
INFORMATION TECHNOLOGY

Information Security (20CS61)
B.TECH VI SEMESTER
R20

INDEX

Experiment – 1

Aim: Implement any two Substitution Techniques using python script.

Algorithm for Substitution Cipher:

Input:

- A String of both lower and upper case letters, called Plaintext.
- An Integer denoting the required key.

Procedure:

- Create a list of all the characters.
- Create a dictionary to store the substitution for all characters.
- For each character, transform the given character as per the rule, depending on whether we're encrypting or decrypting the text.
- Print the new string generated.

Program: 1)

```
import string
# A list containing all characters
all_letters = string.ascii_letters
# create a dictionary to store the substitution for the given alphabet in the plain text based on key
key = 4
dict1 = {all_letters[i]: all_letters[(i+key) % len(all_letters)] for i in range(len(all_letters))}
# Plaintext to be encrypted
plain_txt = "I am studying Data Encryption"
# loop to generate ciphertext
cipher_txt = ''.join([dict1[char] if char in all_letters else char for char in plain_txt])
print("Cipher Text is: ", cipher_txt)
# create a dictionary to store the substitution for the given alphabet in the cipher text based on the key
dict2 = {all_letters[i]: all_letters[(i-key) % len(all_letters)] for i in range(len(all_letters))}
# loop to recover plaintext
decrypt_txt = ''.join([dict2[char] if char in all_letters else char for char in cipher_txt])
print("Recovered plain text: ", decrypt_txt)
```

2)

```
# Python program to demonstrate
# Substitution Cipher
import string
# A list containing all characters
all_letters= string.ascii_letters
# create a dictionary to store the substitution for the given alphabet in the plain text based on the key
dict1 = {}
key = 4
for i in range(len(all_letters)):
    dict1[all_letters[i]] = all_letters[(i+key)%len(all_letters)]
plain_txt= "I am studying Data Encryption"
cipher_txt=[]
# loop to generate ciphertext
for char in plain_txt:
    if char in all_letters:
        temp = dict1[char]
```

```
        cipher_txt.append(temp)
    else:
        temp =char
        cipher_txt.append(temp)
cipher_txt= "".join(cipher_txt)
print("Cipher Text is: ",cipher_txt)
#create a dictionary to store the substitution for the given alphabet in the cipher text based on key
dict2 = {}
for i in range(len(all_letters)):
    dict2[all_letters[i]] = all_letters[(i-key)%(len(all_letters))]
# loop to recover plain text
decrypt_txt = []
for char in cipher_txt:
    if char in all_letters:
        temp = dict2[char]
        decrypt_txt.append(temp)
    else:
        temp = char
        decrypt_txt.append(temp)
decrypt_txt = "".join(decrypt_txt)
print("Recovered plain text :", decrypt_txt)
```

Output:

```
Cipher Text is:  M eq wxyhCmrk Hexe IrgvCtxmsr
Recovered plain text:  I am studying Data Encryption
```

The Playfair Cipher Encryption Algorithm:

The Algorithm consists of 2 steps:

1. **Generate the key Square(5×5):**
   a. The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets). If the plaintext contains J, then it is replaced by I.
   b. The initial alphabets in the key square are the unique alphabets of the key in the order in which they appear followed by the remaining letters of the alphabet in order.
2. **Algorithm to encrypt the plain text:** The plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.

Program:

```
# Python program to implement Playfair Cipher
# Function to convert the string to lowercase
def toLowerCase(text):
    return text.lower()
# Function to remove all spaces in a string
def removeSpaces(text):
    newText = ""
    for i in text:
        if i == " ":
            continue
```

```
        else:
            newText = newText + i
    return newText
# Function to group 2 elements of a string
# as a list element
def Diagraph(text):
    Diagraph = []
    group = 0
    for i in range(2, len(text), 2):
        Diagraph.append(text[group:i])
        group = i
    Diagraph.append(text[group:])
    return Diagraph
 # Function to fill a letter in a string element
# If 2 letters in the same string matches
def FillerLetter(text):
    k = len(text)
    if k % 2 == 0:
        for i in range(0, k, 2):
            if text[i] == text[i+1]:
                new_word = text[0:i+1] + str('x') + text[i+1:]
                new_word = FillerLetter(new_word)
                break
            else:
                new_word = text
    else:
        for i in range(0, k-1, 2):
            if text[i] == text[i+1]:
                new_word = text[0:i+1] + str('x') + text[i+1:]
                new_word = FillerLetter(new_word)
                break
            else:
                new_word = text
    return new_word
list1 = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm',
         'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
# Function to generate the 5x5 key square matrix
def generateKeyTable(word, list1):
    key_letters = []
    for i in word:
        if i not in key_letters:
            key_letters.append(i)
    compElements = []
    for i in key_letters:
        if i not in compElements:
            compElements.append(i)
```

```
    for i in list1:
        if i not in compElements:
            compElements.append(i)
    matrix = []
    while compElements != []:
        matrix.append(compElements[:5])
        compElements = compElements[5:]
    return matrix
def search(mat, element):
    for i in range(5):
        for j in range(5):
            if(mat[i][j] == element):
                return i, j
def encrypt_RowRule(matr, e1r, e1c, e2r, e2c):
    char1 = ''
    if e1c == 4:
        char1 = matr[e1r][0]
    else:
        char1 = matr[e1r][e1c+1]
   char2 = ''
    if e2c == 4:
        char2 = matr[e2r][0]
    else:
        char2 = matr[e2r][e2c+1]
    return char1, char2
def encrypt_ColumnRule(matr, e1r, e1c, e2r, e2c):
    char1 = ''
    if e1r == 4:
        char1 = matr[0][e1c]
    else:
        char1 = matr[e1r+1][e1c]
  char2 = ''
    if e2r == 4:
        char2 = matr[0][e2c]
    else:
        char2 = matr[e2r+1][e2c]
    return char1, char2
def encrypt_RectangleRule(matr, e1r, e1c, e2r, e2c):
    char1 = ''
    char1 = matr[e1r][e2c]
    char2 = ''
    char2 = matr[e2r][e1c]
    return char1, char2

def encryptByPlayfairCipher(Matrix, plainList):
    CipherText = []
```

```
    for i in range(0, len(plainList)):
        c1 = 0
        c2 = 0
        ele1_x, ele1_y = search(Matrix, plainList[i][0])
        ele2_x, ele2_y = search(Matrix, plainList[i][1])

        if ele1_x == ele2_x:
            c1, c2 = encrypt_RowRule(Matrix, ele1_x, ele1_y, ele2_x, ele2_y)
            # Get 2 letter cipherText
        elif ele1_y == ele2_y:
            c1, c2 = encrypt_ColumnRule(Matrix, ele1_x, ele1_y, ele2_x, ele2_y)
        else:
            c1, c2 = encrypt_RectangleRule(
                Matrix, ele1_x, ele1_y, ele2_x, ele2_y)

        cipher = c1 + c2
        CipherText.append(cipher)
    return CipherText


text_Plain = 'instruments'
text_Plain = removeSpaces(toLowerCase(text_Plain))
PlainTextList = Diagraph(FillerLetter(text_Plain))
if len(PlainTextList[-1]) != 2:
    PlainTextList[-1] = PlainTextList[-1]+'z'

key = "Monarchy"
print("Key text:", key)
key = toLowerCase(key)
Matrix = generateKeyTable(key, list1)

print("Plain Text:", text_Plain)
CipherList = encryptByPlayfairCipher(Matrix, PlainTextList)

CipherText = ""
for i in CipherList:
    CipherText += i
print("CipherText:", CipherText)
```

<u>Output:</u>

```
Key text: Monarchy
Plain Text: instruments
CipherText: gatlmzclrqtx
```

Experiment – 2

Aim: Implement any two Transposition Techniques using python script.

Description:

In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.

1. The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.
2. Width of the rows and the permutation of the columns are usually defined by a keyword.
3. For example, the word HACK is of length 4 (so the rows are of length 4), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be “3 1 2 4”.
4. Any spare spaces are filled with nulls or left blank or placed by a character (Example: _).
5. Finally, the message is read off in columns, in the order specified by the keyword.

Encryption

Given text = Geeks for Geeks

Keyword = HACK **Length of Keyword** = 4 (no of rows) **Order of Alphabets in HACK** = 3124

H	A	C	K
3	1	2	4
G	e	e	k
s	_	f	o
r	_	G	e
e	k	s	_

Print Characters of column 1,2,3,4

Encrypted Text = e kefGsGsrekoe_

Decryption

1. To decipher it, the recipient has to work out the column lengths by dividing the message length by the key length.
2. Then, write the message out in columns again, then re-order the columns by reforming the key word.

Program: *(simple code)*

```
import math
key = "HACK"
def encryptMessage(msg):
    k_indx, msg_len = 0, float(len(msg))
    msg_lst = list(msg) + ['_' * int((math.ceil(msg_len / len(key))) * len(key) - msg_len)]
    matrix = [msg_lst[i: i + len(key)] for i in range(0, len(msg_lst), len(key))]
    return ''.join([matrix[j][key.index(sorted(list(key))[k_indx])] for k_indx in range(len(key)) for j in
range(len(matrix))])
def decryptMessage(cipher):
    k_indx, msg_indx, msg_len = 0, 0, float(len(cipher))
    msg_lst = list(cipher)
    key_lst = sorted(list(key))
    dec_cipher = [[None] * len(key) for _ in range(int(math.ceil(msg_len / len(key))))]
    for k_indx in range(len(key)):
        curr_idx = key.index(key_lst[k_indx])
```

```
        for j in range(int(math.ceil(msg_len / len(key)))):
            dec_cipher[j][curr_idx] = msg_lst[msg_indx]
            msg_indx += 1
    try:
        msg = ''.join(sum(dec_cipher, []))
    except TypeError:
        raise TypeError("This program cannot", "handle repeating words.")
    null_count = msg.count('_')
    return msg[: -null_count] if null_count > 0 else msg
msg = "Geeks for Geeks"
cipher = encryptMessage(msg)
print("Encrypted Message: {}".format(cipher))
print("Decrypted Message: {}".format(decryptMessage(cipher)))
```

Output:

```
=============== RESTART: E:/3rd year/VI th sem/IS LAB/2n
d exp.py ===============
Encrypted Message: e  kefGsGsrekoe_
Decrypted Message: Geeks for Geeks
```

Given a plain-text message and a numeric key, cipher/de-cipher the given text using Rail Fence algorithm.

The rail fence cipher (also called a zigzag cipher) is a form of transposition cipher. It derives its name from the way in which it is encoded.

Examples:

Encryption

Input : "GeeksforGeeks "

Key = 3

Output : GsGsekfrek eoe

Decryption

Input : GsGsekfrek eoe

Key = 3

Output : "GeeksforGeeks "

2) Encryption

Input : "defend the east wall"

Key = 3

Output : dnhaweedtees alf tl

Decryption

Input : dnhaweedtees alf tl

Key = 3

Output : defend the east wall

3) Encryption

Input : "attack at once"

Key = 2

Output : atc toctaka ne

Decryption

Input : "atc toctaka ne"

Key = 2 Output : attack at once

Program-2:

```
import math
key = "HACK"
# Encryption
def encryptMessage(msg):
    cipher = ""
    # track key indices
    k_indx = 0
    msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))
    # calculate column of the matrix
    col = len(key)
    # calculate maximum row of the matrix
    row = int(math.ceil(msg_len / col))
    # add the padding character '_' in empty
    # the empty cell of the matix
    fill_null = int((row * col) - msg_len)
    msg_lst.extend('_' * fill_null)
    # create Matrix and insert message and
    # padding characters row-wise
    matrix = [msg_lst[i: i + col]
              for i in range(0, len(msg_lst), col)]
    # read matrix column-wise using key
    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
# Decryption
def decryptMessage(cipher):
    msg = ""
     # track key indices
    k_indx = 0
    # track msg indices
    msg_indx = 0
    msg_len = float(len(cipher))
    msg_lst = list(cipher)
    # calculate column of the matrix
    col = len(key)
    # calculate maximum row of the matrix
    row = int(math.ceil(msg_len / col))
    # convert key into list and sort
    # alphabetically so we can access
    # each character by its alphabetical position.
```

```
    key_lst = sorted(list(key))
    # create an empty matrix to
    # store deciphered message
    dec_cipher = []
    for _ in range(row):
        dec_cipher += [[None] * col]
    # Arrange the matrix column wise according
    # to permutation order by adding into new matrix
    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
    # convert decrypted msg matrix into a string
    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
# Driver Code
msg = "I am studying Data Encryption"
cipher = encryptMessage(msg)
print("Encrypted Message: {}".    format(cipher))
print("Decryped Message: {}".
       format(decryptMessage(cipher)))
```

Output:

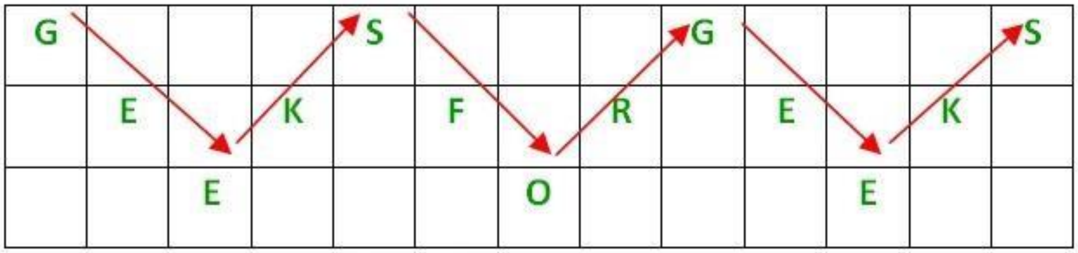
```
Encrypted Message:  sy act_atiD ri_I dgtnpnmunaEyo_
Decryped Message: I am studying Data Encryption
```

Encryption

In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.

- In the rail fence cipher, the plain-text is written downwards and diagonally on successive rails of an imaginary fence.
- When we reach the bottom rail, we traverse upwards moving diagonally, after reaching the top rail, the direction is changed again. Thus the alphabets of the message are written in a zig-zag manner.
- After each alphabet has been written, the individual rows are combined to obtain the cipher-text.

For example, if the message is “GeeksforGeeks” and the number of rails = 3 then cipher is prepared as:



.’.Its encryption will be done row wise i.e. GSGSEKFREKEOE

Decryption

As we’ve seen earlier, the number of columns in rail fence cipher remains equal to the length of plain-text message. And the key corresponds to the number of rails.

- Hence, rail matrix can be constructed accordingly. Once we’ve got the matrix we can figure-out the spots where texts should be placed (using the same way of moving diagonally up and down alternatively).
- Then, we fill the cipher-text row wise. After filling it, we traverse the matrix in zig-zag manner to obtain the original text.

Implementation:

Let cipher-text = “GsGsekfrek eoe” , and Key = 3

- Number of columns in matrix = len(cipher-text) = 13
- Number of rows = key = 3

Hence original matrix will be of 3*13 , now marking places with text as ‘*’ we get

```
 * _ _ _ * _ _ _ * _ _ _ *
_ * _ * _ * _ * _ * _ *
_ _ * _ _ _ * _ _ _ * _
```

Program:

```
def rail_fence(text, key, mode):
    if mode == "encrypt":
        rail = [""] * key
        index = 0
        for i in range(len(text)):
            rail[index] += text[i]
            if index == key - 1:
                direction = -1
            elif index == 0:
                direction = 1
            index += direction
        return "".join(rail)
    elif mode == "decrypt":
        rail = [""] * key
        index = 0
        for i in range(len(text)):
```

```
            rail[index] += "*"
            if index == key - 1:
                direction = -1
            elif index == 0:
                direction = 1
            index += direction
        text_index = 0
        for i in range(key):
            for j in range(len(rail[i])):
                if rail[i][j] == "*":
                    rail[i] = rail[i][:j] + text[text_index] + rail[i][j+1:]
                    text_index += 1
        index = 0
        plain_text = ""
        for i in range(len(text)):
            plain_text += rail[index][0]
            rail[index] = rail[index][1:]
            if index == key - 1:
                direction = -1
            elif index == 0:
                direction = 1
            index += direction
        return plain_text
    else:
        return "Invalid mode. Mode must be either 'encrypt' or 'decrypt'."

text = input("Enter the input string: ")
key = int(input("Enter the key: "))
encrypted_text = rail_fence(text, key, "encrypt")
print("Encrypted text:", encrypted_text)
decrypted_text = rail_fence(encrypted_text, key, "decrypt")
print("Decrypted text:", decrypted_text)
```

Output:

```
============== RESTART: E:/3rd year/VI th sem/IS LAB/2ND exp_b.py
Enter the input string: HELLO WORLD
Enter the key: 3
Encrypted text: HOREL OLLWD
Decrypted text: HELLO WORLD
```

Experiment – 3

Aim: Implement any two Symmetric algorithms using python script.

1) Data Encryption Standard (DES).
2) RSA encryption algorithm.

Description:

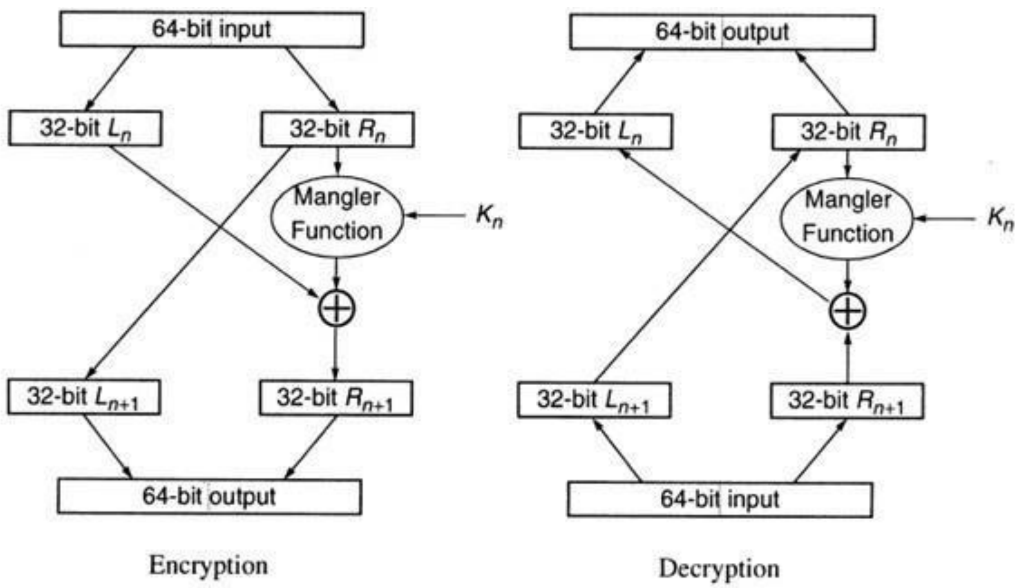
DES is a symmetric encryption system

DES is a symmetric encryption system that uses 64-bit blocks, 8 bits of which areused for parity checks. The key therefore has a "useful" length of 56 bits, which means thatonly 56 bits are actually used in the algorithm. The algorithm involves carrying out combinations, substitutions and permutations between the text to be encrypted and the key, while making sure the operations can be performed in both directions. The key is ciphered on64 bits and made of 16 blocks of 4 bits, generally denoted k_1 to k_{16}. Given that "only" 56 bits are actually used for encrypting, there can be 2^{56} different keys.

The main parts of the algorithm are as follows:

- Fractioning of the text into 64-bit blocks
- Initial permutation of blocks
- Breakdown of the blocks into two parts: left and right, named L and R
- Permutation and substitution steps repeated 16 times
- Re-joining of the left and right parts then inverse initial permutation

Example:



Algorithm:

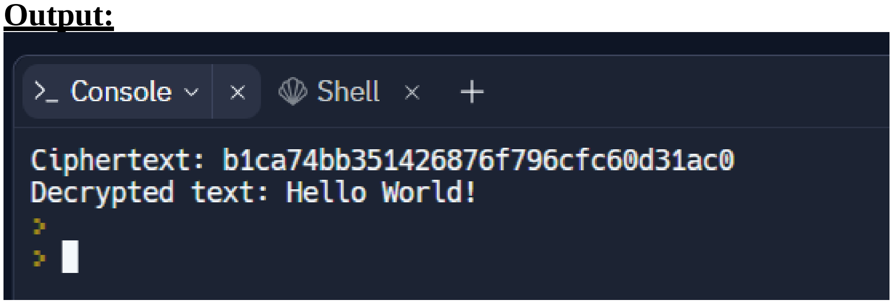
STEP-1: Read the 64-bit plain text.
STEP-2: Split it into two 32-bit blocks and store it in two different arrays.
STEP-3: Perform XOR operation between these two arrays.
STEP-4: The output obtained is stored as the second 32-bit sequence and the originalsecond 32-bit sequence forms the first part.
STEP-5: Thus the encrypted 64-bit cipher text is obtained in this way. Repeat the sameprocess for the remaining plain text characters.

Program:

```
from Crypto.Cipher import DES
```

```
from Crypto.Util.Padding import pad, unpad
def des_encrypt(plaintext, key):
    # Convert the key to bytes
    key = bytes.fromhex(key)
    # Create a DES cipher object and encrypt the plaintext
    cipher = DES.new(key, DES.MODE_ECB)
    ciphertext = cipher.encrypt(pad(plaintext.encode('utf-8'), DES.block_size))
    # Convert the ciphertext to hexadecimal and return as string
    return ciphertext.hex()
def des_decrypt(ciphertext, key):
    # Convert the key to bytes
    key = bytes.fromhex(key)
    # Create a DES cipher object and decrypt the ciphertext
    cipher = DES.new(key, DES.MODE_ECB)
    decryptedtext = unpad(cipher.decrypt(bytes.fromhex(ciphertext)), DES.block_size)
    # Convert the decryptedtext to string and return
    return decryptedtext.decode('utf-8')
plaintext = "Hello World!"
key = "133457799BBCDFF1"
# Encrypt the plaintext using DES with the given key
ciphertext = des_encrypt(plaintext, key)
print("Ciphertext:", ciphertext)
# Decrypt the ciphertext using DES with the given key
decryptedtext = des_decrypt(ciphertext, key)
print("Decrypted text:", decryptedtext)
```

Output:



Description:

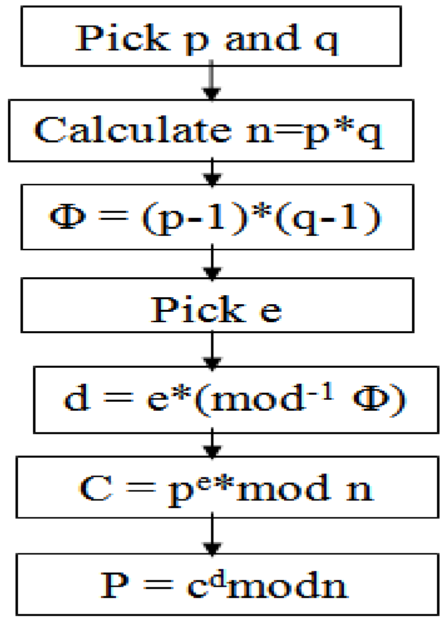
RSA(Rivest-Shamir-Adleman) encryption algorithm

RSA is an algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography, because one of them can be given toeveryone. A basic principle behind RSA is the observation that it is practical to find threevery large positive integers e, d and n such that with modular exponentiation for all integer m:

$$(m^e)^d = m \pmod{n}$$

The public key is represented by the integers n and e; and, the private key, by the integer d. m represents the message. RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount oftime using the private key.

Example:



Algorithm:

The RSA algorithm is a widely used public-key encryption algorithm named after its inventors Ron Rivest, Adi Shamir, and Leonard Adleman. It is based on the mathematical concepts of prime factorization and modular arithmetic.

The algorithm for RSA is as follows:

1. Select 2 prime numbers, preferably large, p and q.
2. Calculate n = p*q.
3. Calculate phi(n) = (p-1)*(q-1)
4. Choose a value of e such that 1<e<phi(n) and gcd(phi(n), e) = 1.
5. Calculate d such that d = (e^-1) mod phi(n).

Here the public key is {e, n} and private key is {d, n}. If M is the plain text then the cipher text C = (M^e) mod n. This is how data is encrypted in RSA algorithm. Similarly, for decryption, the plain text M = (C^d) mod n.

Example: Let p=3 and q=11 (both are prime numbers).

- Now, n = p*q = 3*11 = 33
- phi(n) = (p-1)*(q-1) = (3-1)*(11-1) = 2*10 = 20
- Value of e can be 7 since 1<7<20 and gcd(20, 7) = 1.
- Calculating d = 7^-1 mod 20 = 3.
- Therefore, public key = {7, 33} and private key = {3, 33}.

Suppose our message is M=31. You can encrypt and decrypt it using the RSA algorithm as follows:

Encryption: C = (M^e) mod n = 31^7 mod 33 = 4

Decryption: M = (C^d) mod n = 4^3 mod 33 = 31

Since we got the original message that is plain text back after decryption, we can say that the algorithm worked correctly.

Program:

```
import math
# step 1
p = 3
```

```
q = 7
# step 2
n = p*q
print("n =", n)
# step 3
phi = (p-1)*(q-1)
# step 4
e = 2
while(e<phi):
    if (math.gcd(e, phi) == 1):
        break
    else:
        e += 1
print("e =", e)
# step 5
k = 2
d = ((k*phi)+1)/e
print("d =", d)
print(f'Public key: {e, n}')
print(f'Private key: {d, n}')

# plain text
msg = 11
print(f'Original message:{msg}')

# encryption
C = pow(msg, e)
C = math.fmod(C, n)
print(f'Encrypted message: {C}')

#  decryption
M = pow(C, d)
M = math.fmod(M, n)

print(f'Decrypted message: {M}')
```

Output:

```
=============== RESTART: E:/3rd year/VI th sem/IS LAB/Exp3_RSA.py =
n = 21
e = 5
d = 5.0
Public key: (5, 21)
Private key: (5.0, 21)
Original message:11
Encrypted message: 2.0
Decrypted message: 11.0
```

Experiment – 4

Aim: Implement any two Private -Key based algorithms using python script.

1) Triple DES (3DES).
2) AES (Advanced Encryption Standard).

Description:

Private-key based algorithms in Information Security are also known as symmetric key algorithms. They use a single key for both encryption and decryption, and the same key must be kept secret by both the sender and the receiver in order to maintain the security of the communication. Here are some commonly used private-key based algorithms in IS:

1) **Advanced Encryption Standard (AES)**

   AES is a widely used symmetric key algorithm that uses a block cipher with a block size of 128 bits and a key size of 128, 192, or 256 bits. It is known for its speed and resistance to attacks, and is widely used in applications such as secure communication and data storage.

2) **Triple DES (3DES)**

   3DES is a variant of DES that uses three rounds of encryption with three different keys. It uses a block size of 64 bits and a key size of 168 bits (three 56-bit keys). While 3DES is more secure than DES, it is slower and less efficient than AES.

Program:

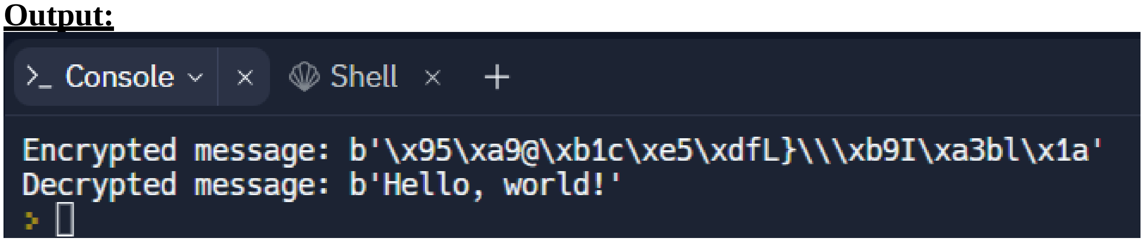
AES Encryption and Decryption

AES (Advanced Encryption Standard) is a symmetric key encryption algorithm that is widely used to protect data. Here is an implementation of AES encryption and decryption using Python:

Code:

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
# Generate random key and IV
key = b'secretkey1234567'
iv = b'iv12345678901234'

# Encrypt  message
message = b"Hello, world!"
cipher = AES.new(key, AES.MODE_CBC, iv)
ciphertext = cipher.encrypt(pad(message, AES.block_size))

# Decrypt message
cipher = AES.new(key, AES.MODE_CBC, iv)
plaintext = unpad(cipher.decrypt(ciphertext), AES.block_size)

print("Encrypted message:", ciphertext)
print("Decrypted message:", plaintext)
```

Output:



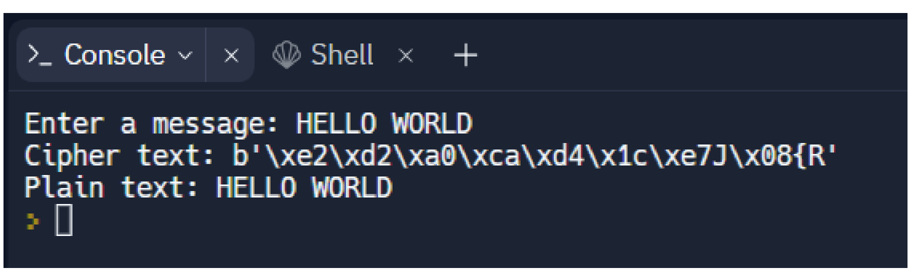
Triple DES or 3 DES Encryption and Decryption

DES (Data Encryption Standard) is a symmetric key encryption algorithm that is widely used to protect data. Here is an implementation of DES encryption and decryption using

Python Code:

```
from Crypto.Cipher import DES3
from Crypto.Random import get_random_bytes

while True:
    try:
        key = DES3.adjust_key_parity(get_random_bytes(24))
        break
    except ValueError:
        pass

def encrypt(msg):
    cipher = DES3.new(key, DES3.MODE_EAX)
    nonce = cipher.nonce
    ciphertext = cipher.encrypt(msg.encode('ascii'))
    return nonce, ciphertext

def decrypt(nonce, ciphertext):
    cipher = DES3.new(key, DES3.MODE_EAX, nonce=nonce)
    plaintext = cipher.decrypt(ciphertext)
    return plaintext.decode('ascii')

nonce, ciphertext = encrypt(input('Enter a message: '))
plaintext = decrypt(nonce, ciphertext)
print(f'Cipher text: {ciphertext}')
print(f'Plain text: {plaintext}')
```

Output:



Experiment – 5

Aim : Explore any four network diagnosis tools.

Description:

Network diagnosis tools are software applications or hardware devices that are used to identify and troubleshoot problems on a computer network. These tools are designed to help network administrators and IT professionals diagnose network issues, identify network threats, and optimize network performance.

Network diagnosis tools can take many forms, ranging from simple command-line utilities to complex graphical user interfaces. Some common types of network diagnosis tools include:

1. Ping and Traceroute: These command-line utilities are used to test network connectivity and identify network routing issues.
2. Network protocol analyzers: These tools capture and display network traffic in real-time, allowing network administrators to troubleshoot network protocols and identify network threats.
3. Network scanners: These tools scan a network for open ports and vulnerable systems, allowing administrators to identify potential security risks.
4. Performance monitoring tools: These tools monitor network traffic and performance metrics, allowing administrators to identify performance bottlenecks and optimize network performance.
5. Configuration management tools: These tools are used to manage and automate network device configurations, allowing administrators to enforce network policies and ensure consistent network configuration.

Overall, network diagnosis tools are essential for maintaining the health and security of computer networks. By using these tools to identify and troubleshoot network issues, administrators can minimize downtime, improve network performance, and ensure the integrity of network data.

Tools:

1. Ping Tools

The ICMP ping tool is a basic network troubleshooting tool that lets you assess if a device is reachable on the network. It reports on errors such as packet loss, round-trip-time, etc.

The usual ping requests are based on the ICMP echo request protocol. There are other variations of ping requests such as SNMP ping and proxy ping.

SNMP ping: It is used to check if the simple network management protocol (SNMP) is enabled in a network device. If SNMP is enabled, the device responds with a set of basic information such as DNS name, system name, location, system type, system description, etc.

Ping Statistics for 192.[redacted]

DNS Name	[redacted]
IP Address	192.[redacted]
Packet Count	1 Packets
Packet Size	56 bytes
Time to Live	255 seconds/hops
Timed out	4 Seconds
Packet Sent	1 Packets
Packet Received	1 Packets
Packet Loss	0 0% loss

Round Trip Time

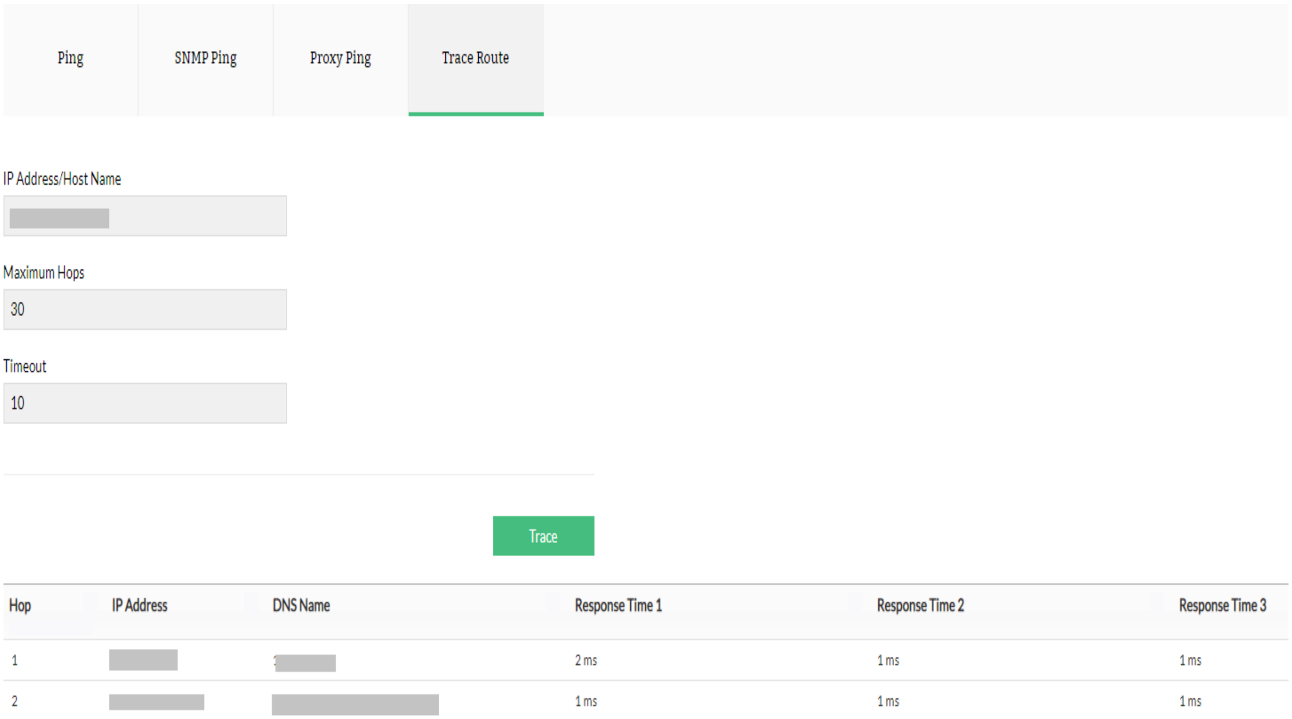
Maximum	2 ms
Minimum	2 ms
Average	2 ms

Proxy ping: This is used to ping a destination device behind a proxy. Basically, the pinging device sends an SNMP SET command to the proxy router to send an ICMP echo request to the destination device. The response is collected by the proxy device. This response is fetched using the SNMP GET command. This ping also requires SNMP to be enabled in the proxy device with the write community string enabled.

These ping commands are useful to diagnose IP problems and network connectivity issues that could be due to faulty interfaces, LAN issues, unavailable ports, configuration issues, etc., and are mostly used in combination with the traceroute network troubleshooting utility.

2. Tracert/ Trace Route

Tracert (Windows) or traceroute (Linux) is a network diagnostic and troubleshooting tool to view the route and measure transit delays of data packets in a network. It displays the number of hops between the source and destination devices based on the hop limit concept, modifying the Time To Live (TTL) values.



A traceroute tool is useful to identify response delays (high latency), routing loops and points of failure or packet loss in a network.

Traceroute is a command-line tool that is used to identify the route taken by packets as they travel between two devices on a network. It sends a series of packets to the target device with increasing time-to-live (TTL) values. Each packet is then returned by the next device on the route, along with the time taken for the packet to travel between the devices. Traceroute is commonly used to diagnose network routing issues and to identify the location of network congestion.

3 .Netsat

Netstat is a command-line tool that is used to display network connection statistics for a device. It shows active connections, listening ports, and network interface statistics, among other things. Netstat is commonly used to diagnose network connectivity issues and to identify suspicious network activity.

The netstat command is a highly practical tool for network diagnostics, configurations, and other port-scanning activities. More specifically, system administrators use it for network troubleshooting and performance diagnostics.

The netstat command works on Microsoft Windows, Linux, Unix, FreeBSD, and more. Therefore, all the commands in this article will produce the same results irrespective of your operating system, unless otherwise stated for Linux.

The Linux operating system comes with a considerable number of built-in capabilities pre-installed. Depending on their level of expertise, users may not be fully aware of the capabilities of a particular command. This article provides the basics of netstat and how to troubleshoot network issues with it

Functions

We will learn how the netstat command functions by seeing its commonly used applications. We will see how to generate routing information, network interface statistics, or run port-scanning operations with the command. It might be a good idea to take notes on the most frequently recurring options and what they do, because they will come in handy while working with other commands.

Displaying kernel routing table

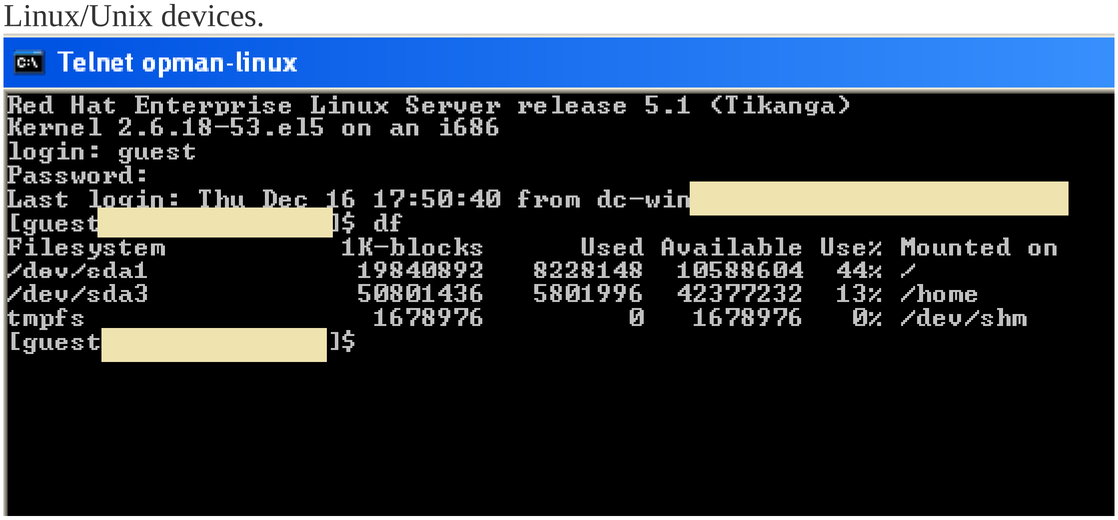
Using the netstat command with the -r option lists the kernel routing information in the same way as with the route command.

```
$ netstat -rn
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS
Window  irtt Iface
0.0.0.0         192.168.1.1     0.0.0.0         UG        0 0
0 eth0
192.168.1.0     0.0.0.0         255.255.255.0   U         0 0
0 eth0
```

Note that the additional -n option is used to disable hostname lookup. It configures the netstat command to display addresses as dot-separated quad IP numbers instead of host and network names in the form of symbols.

4. Telnet/ SSH

Telnet or Secure Shell (SSH) utility allows you to troubleshoot issues by establishing a CLI session with Linux/Unix devices.



It is a simple yet effective network troubleshooting tool that enables you to act on any alert by executing CLI commands to remediate L1/L2 network problems.

Experiment – 6

1. **Aim:** Study of packet sniffer tools like wireshark, ethereal, tcpdump etc

2. **Objectives:** To observe the performance in promiscuous & non-promiscuous mode &
3. **Outcomes:** The learner will be able to:-

to find the packets based on different filters.

- Identify different packets moving in/out of network using packet sniffer for network analysis.
- Understand professional, ethical, legal, security and social issues and responsibilities. Also will be able to analyze the local and global impact of computing on individuals, organizations, and society.
- Match the industry requirements in the domains of Database management, Programming and Networking with the required management skills.

4. **Hardware / Software Required:** Wireshark, Ethereal and tcpdump.

5. **Theory:**

Wireshark, a network analysis tool formerly known as Ethereal, captures packets in real time and display them in human-readable format. Wireshark includes filters, color-coding and other features that let you dig deep into network traffic and inspect individual packets.

Applications:

Features:

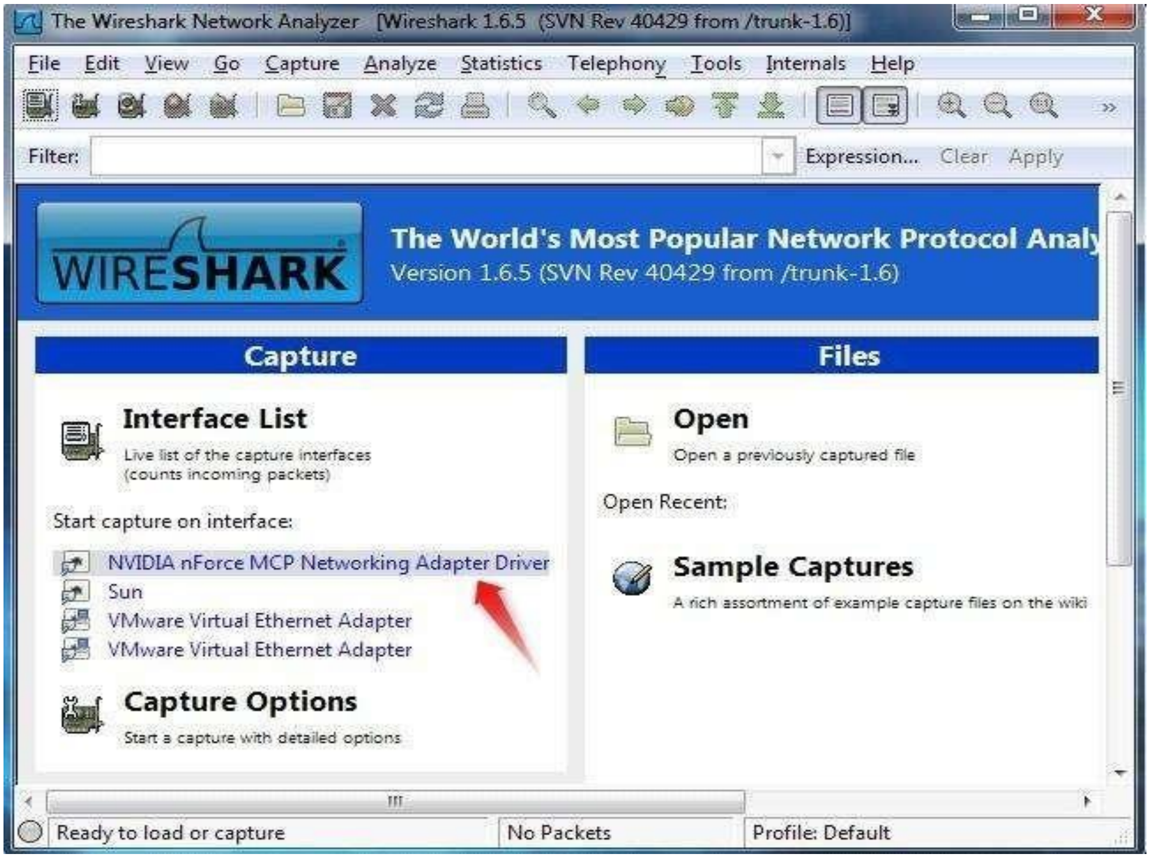
- Network administrators use it to troubleshoot network problems
- Network security engineers use it to examine security problems
- Developers use it to debug protocol implementations
- People use it to learn network protocol internals beside these examples can be helpful in many other situations too.

The following are some of the many features wireshark provides:

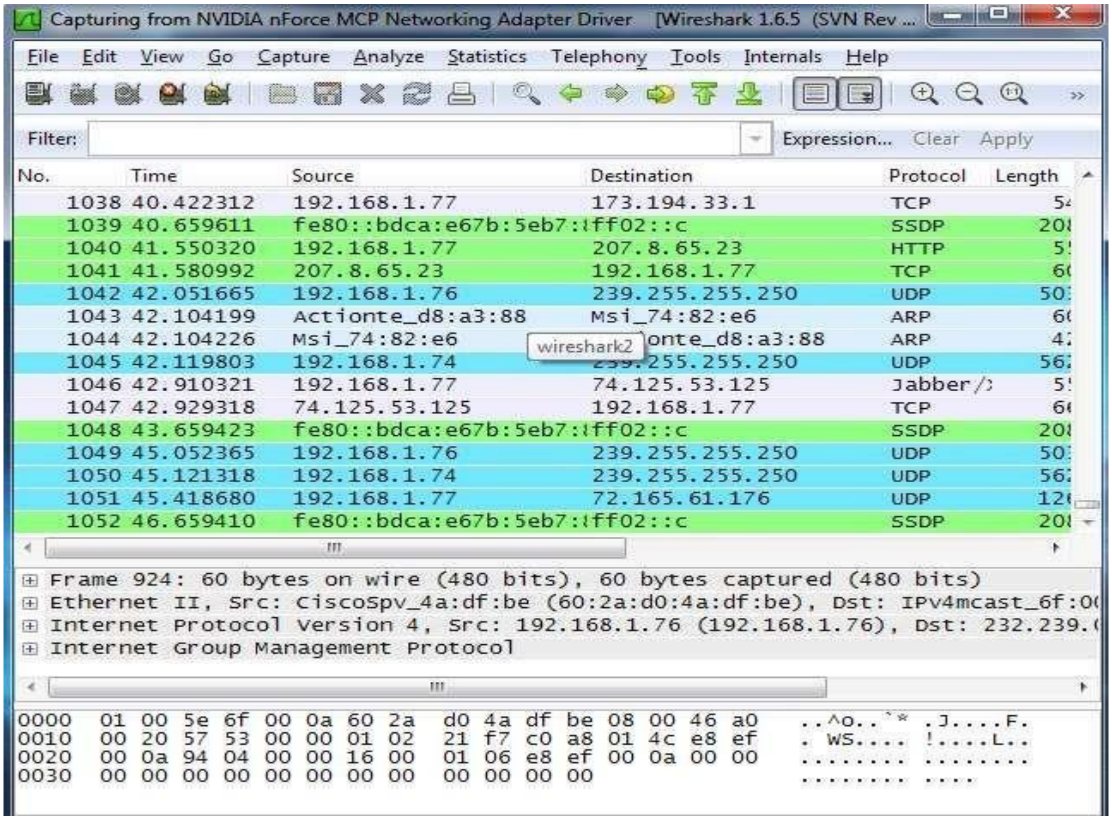
- and a number of other packet capture programs.
- Capture live packet data from a network interface.
- Open files containing packet data captured with tcpdump/WinDump, Wireshark,
  Import packets from text files containing hex dumps of packet data.
  Available for UNIX and Windows.
-
- Display packets with very detailed protocol information.
- Export some or all packets in a number of capture file formats.
- Filter packets on many criteria.
- Search for packets on many criteria.
- Colorize packet display based on filters.
- Create various statistics.

Capturing Packets

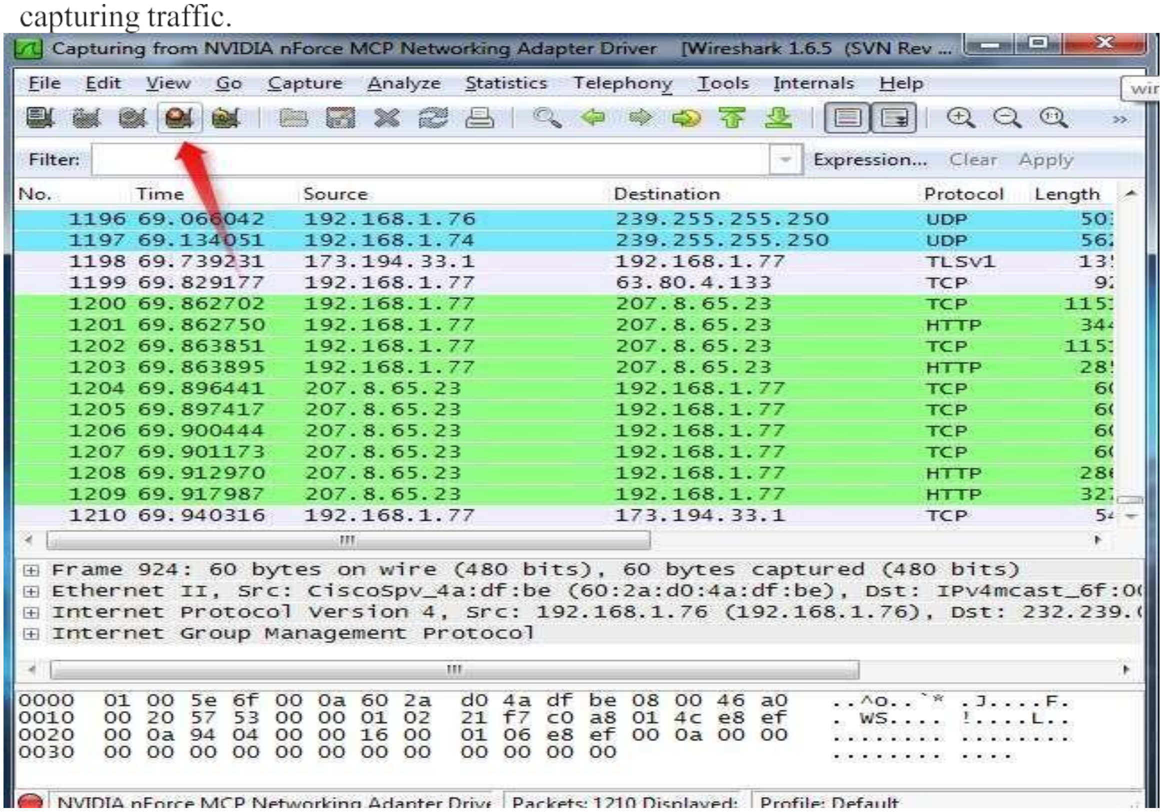
After downloading and installing wireshark, you can launch it and click the name of an interface under Interface List to start capturing packets on that interface. For example, if you want to capture traffic on the wireless network, click your wireless interface. You can configure advanced features by clicking Capture Options.



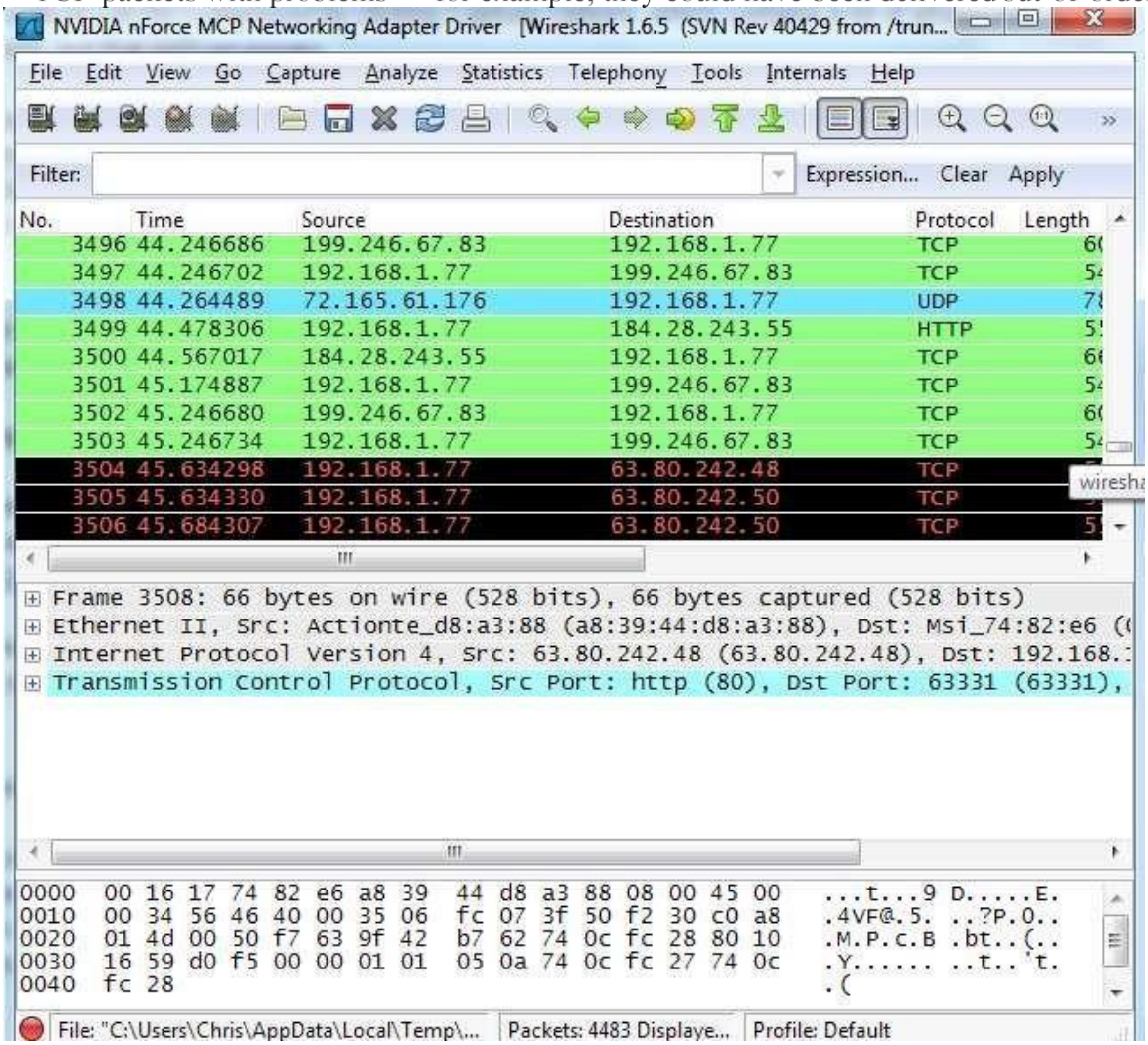
As soon as you click the interface's name, you'll see the packets start to appear in real time. Wireshark captures each packet sent to or from your system. If you're capturing on a wireless interface and have promiscuous mode enabled in your capture options, you'll also see other the other packets on the network.



Click the stop capture button near the top left corner of the window when you want to stop capturing traffic.



Wireshark uses colors to help you identify the types of traffic at a glance. By default, green is TCP traffic, dark blue is DNS traffic, light blue is UDP traffic, and black identifies TCP packets with problems — for example, they could have been delivered out-of-order.



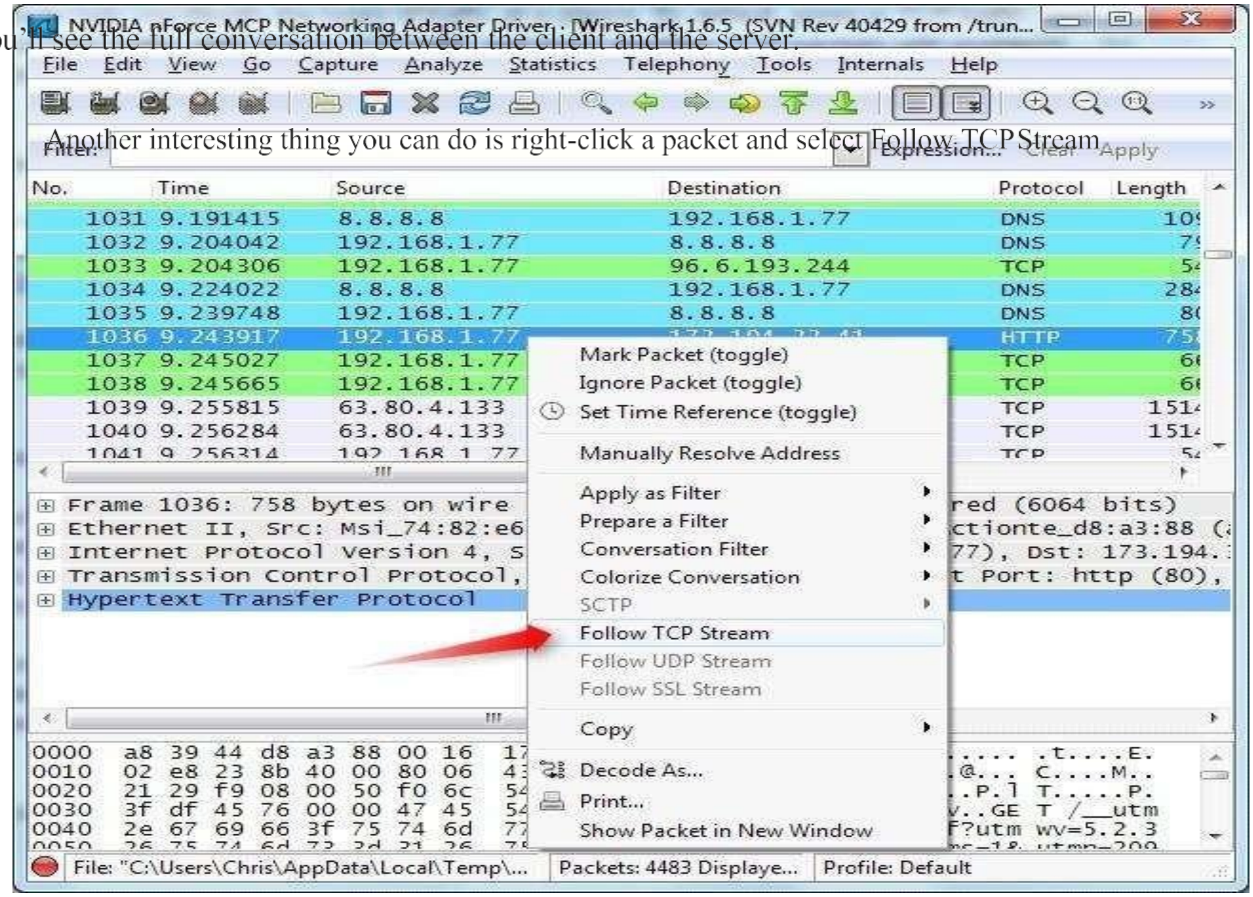
Filtering Packets

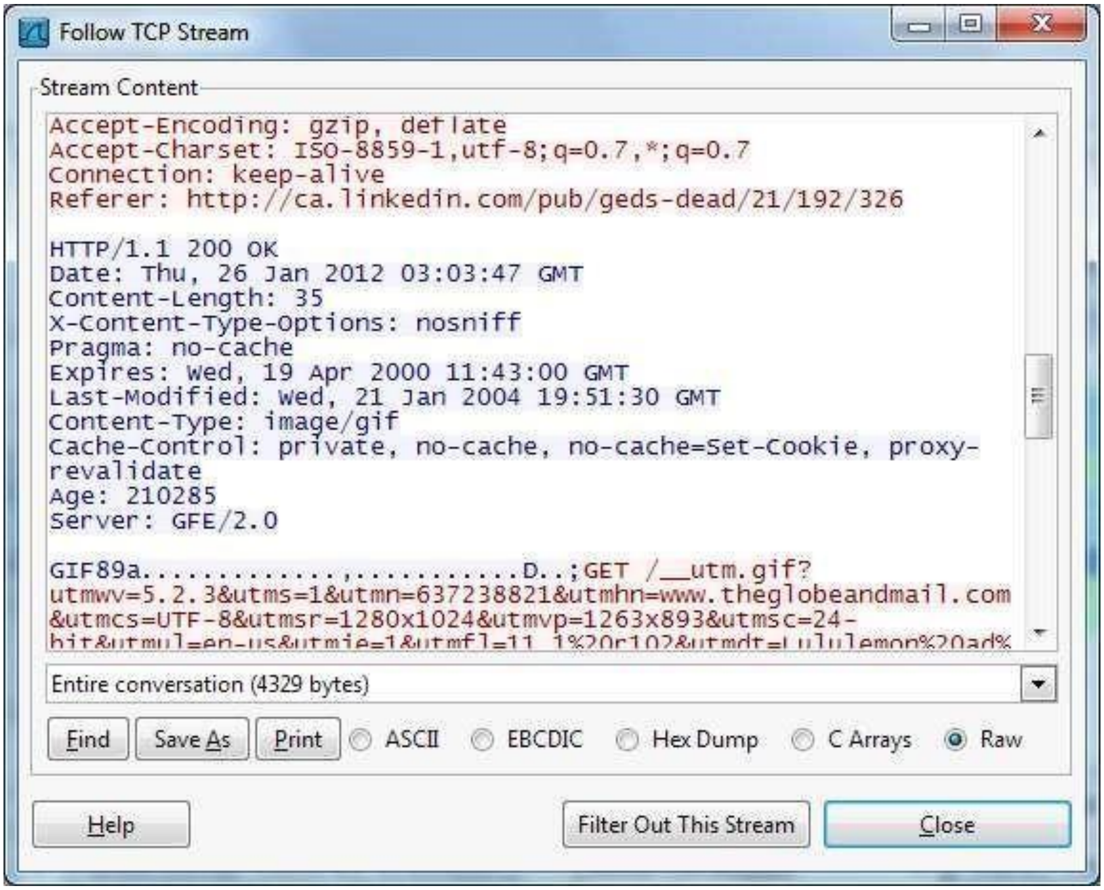
If you're trying to inspect something specific, such as the traffic a program sends when phoning home, it helps to close down all other applications using the network so you can narrow down the traffic. Still, you'll likely have a large amount of packets to sift through. That's where Wireshark's filters come in.

The most basic way to apply a filter is by typing it into the filter box at the top of the window and clicking Apply (or pressing Enter). For example, type "dns" and you'll see only DNS packets. When you start typing, Wireshark will help you autocomplete your filter.

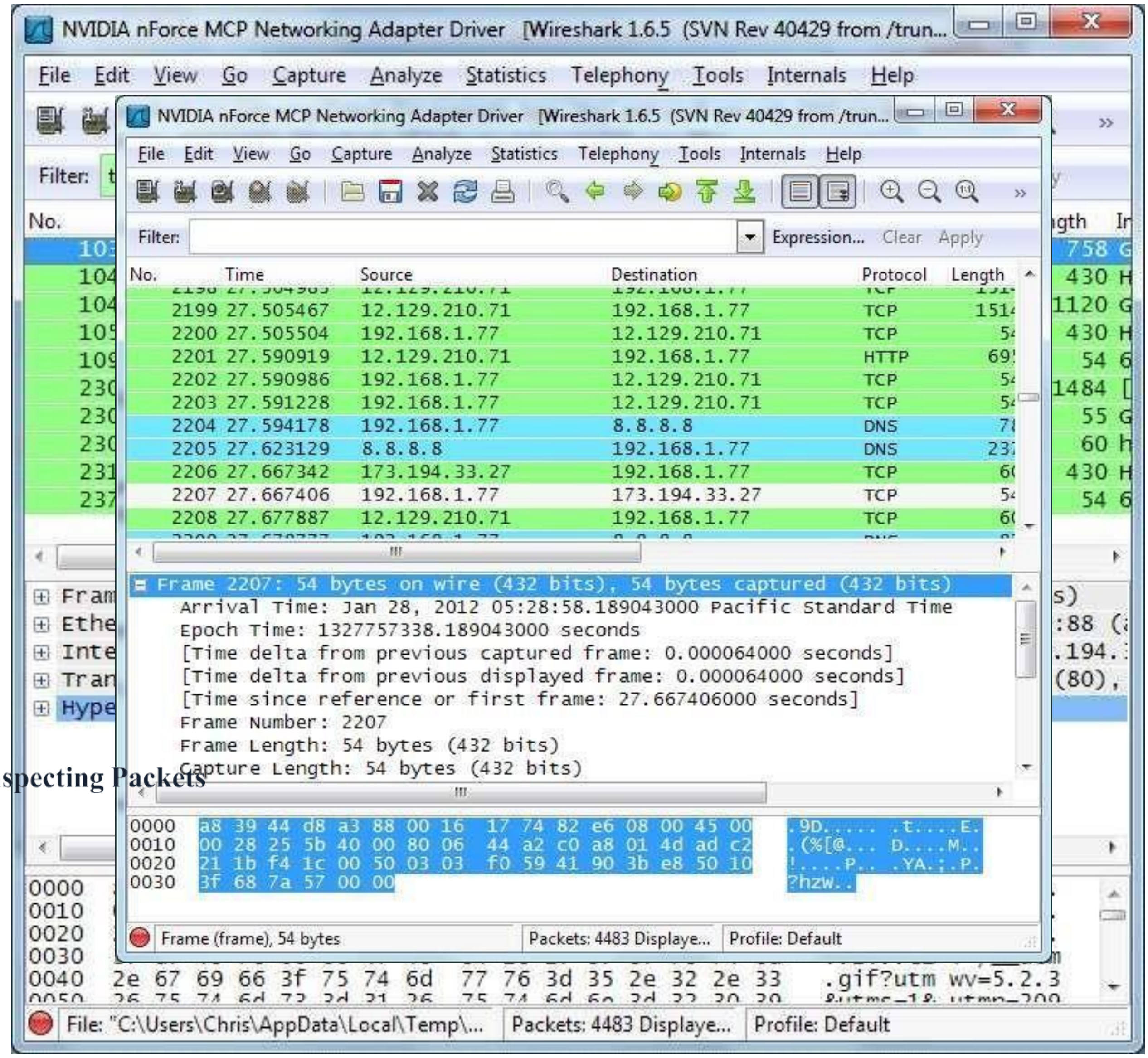
You'll see the full conversation between the client and the server.

Another interesting thing you can do is right-click a packet and select Follow TCP Stream.





Close the window and you'll find a filter has been applied automatically — Wireshark is showing you the packets that make up the conversation.



Inspecting Packets

Click a packet to select it and you can dig down to view its details.

Experiment – 7

1. **Aim:** Download and install nmap. Use it with different options to scan open
2. **Objectives:** objective of this module to learn nmap installation & use this to scan ports,perform OS fingerprinting, do a ping scan, tcp port scan, udp port scan, etc. different ports.

3. **Outcomes:** The learner will be able to:-

   - Scan the network using scanning techniques available in NMAP.
   - Use current techniques, skills, and tools necessary for computing practice

4. **Hardware / Software Required :** NMAP Tool
5. **Theory:**

Nmap (Network Mapper) is a security scanner originally written by Gordon Lyon (also known by his pseudonym Fyodor Vaskovich) used to discover hosts and services on a computer network, thus creating a "map" of the network. To accomplish its goal, Nmap sends specially crafted packets to the target host and then analyzes the responses. Unlike many simple port scanners that just send packets at some predefined constant rate, Nmap accounts for the network

conditions (latency fluctuations, network congestion, the target interference with the scan) during the run. Also, owing to the large and active user community providing feedback and contributing to its features, Nmap has been able to extend its discovery capabilities beyond simply figuring out whether a host is up or down and which ports are open and closed; it can determine the operating system of the target, names and versions of the listening services, estimated uptime,type of device, and presence of a firewall.

Nmap features include: characteristics of network devices.

Host Discovery – Identifying hosts on a network. For example, listing the hosts which respond to pings or have a particular port open.

Version Detection – Interrogating listening network services listening on remote devices

Port Scanning – Enumerating the open ports on one or more target hosts.

OS Detection – Remotely determining the operating system and some hardware to determine the application name and version number.

Basic commands working in Nmap

For target specifications:

nmap <target's URL or IP with spaces between them>

For OS detection:

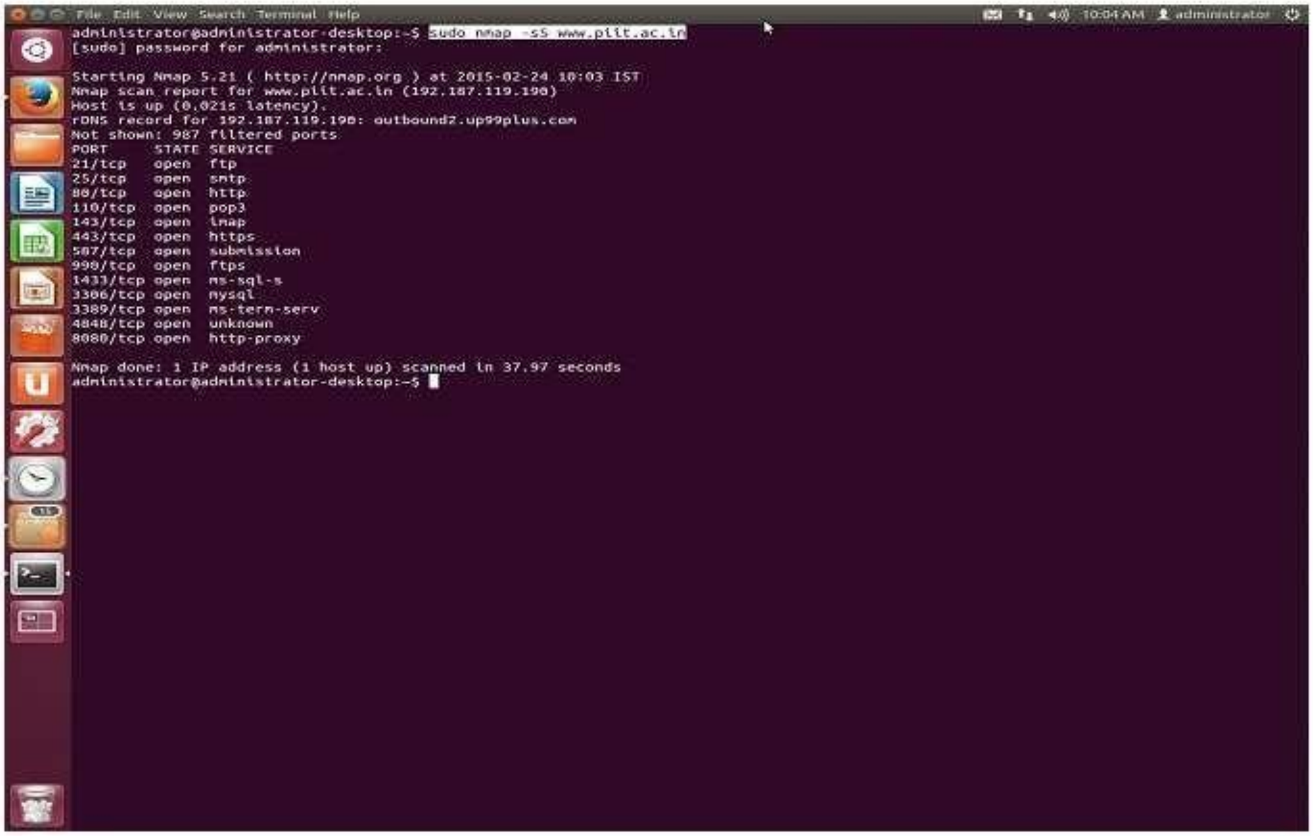
nmap -O <target-host's URL or IP>

For version detection:

nmap -sV <target-host's URL or IP>

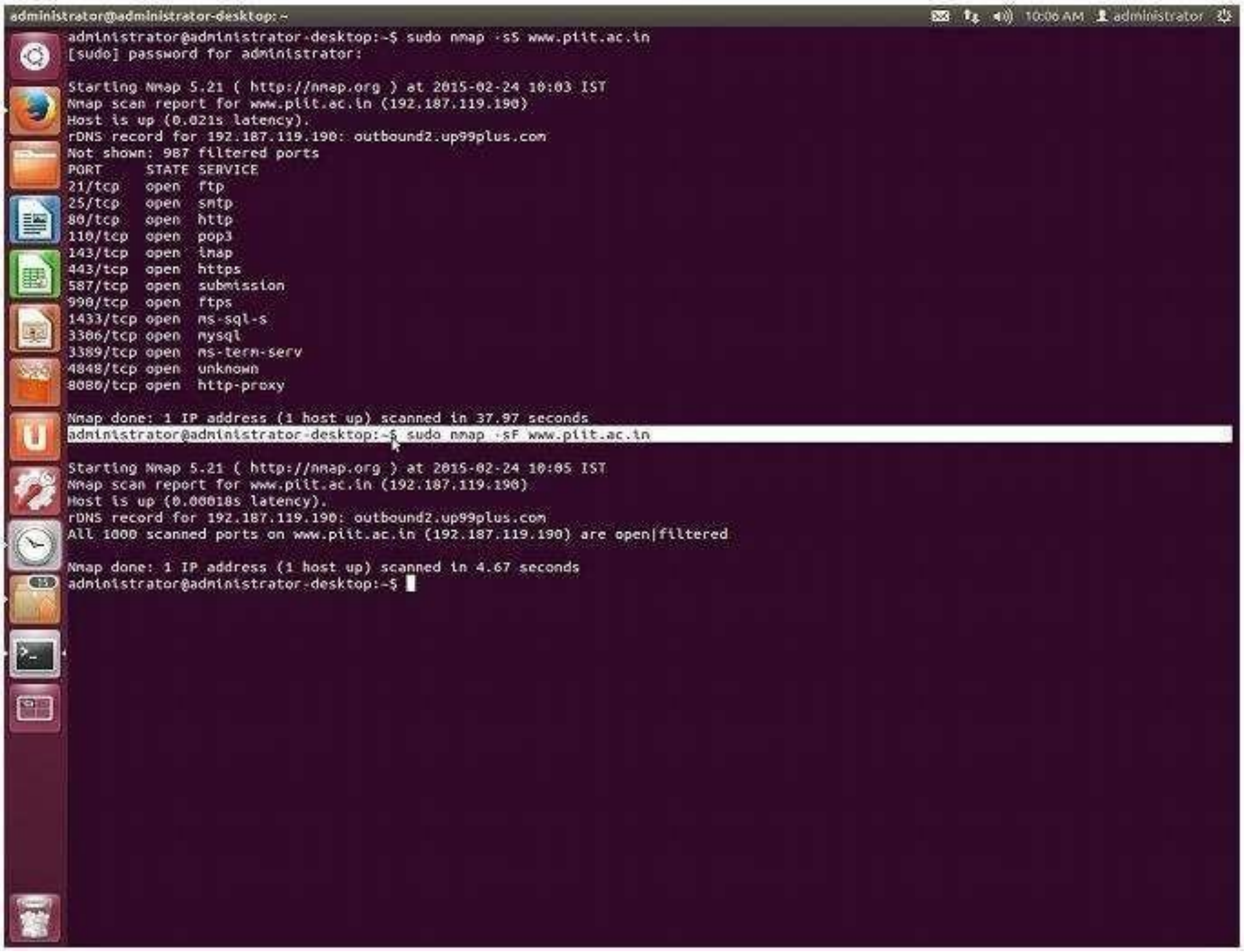
After the installation of nmap:> **sudo apt-get install nmap**

SYN scan is the default and most popular scan option for good reasons. It can be performed quickly, scanning thousands of ports per second on a fast network not hampered by restrictive firewalls. It is also relatively unobtrusive and stealthy since it never completes TCP connections.

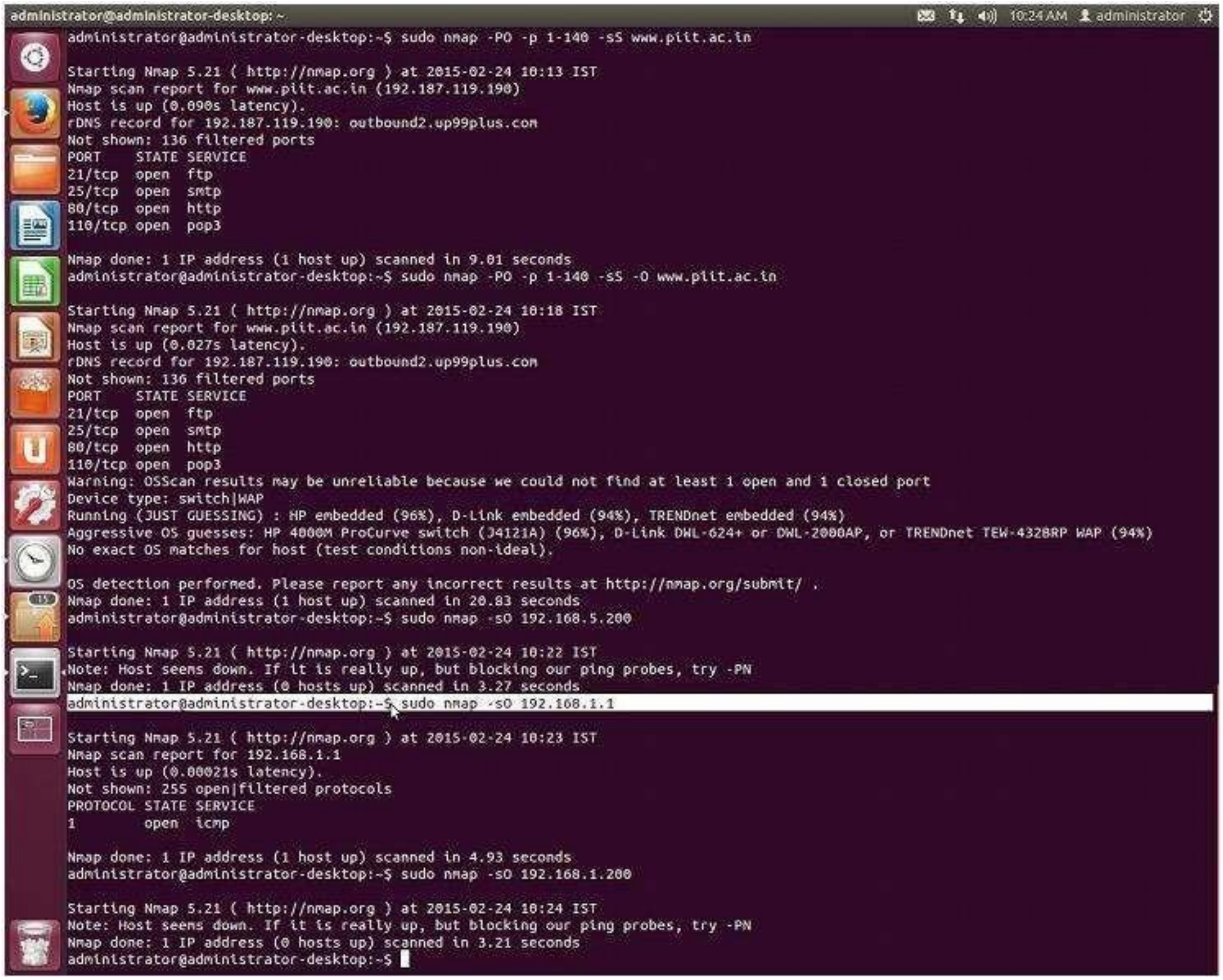


FIN scan (-sF)

Sets just the TCP FIN bit.



-sV (Version detection) :Enables version detection, as discussed above. Alternatively, we can use -A, which enables version detection among other things.

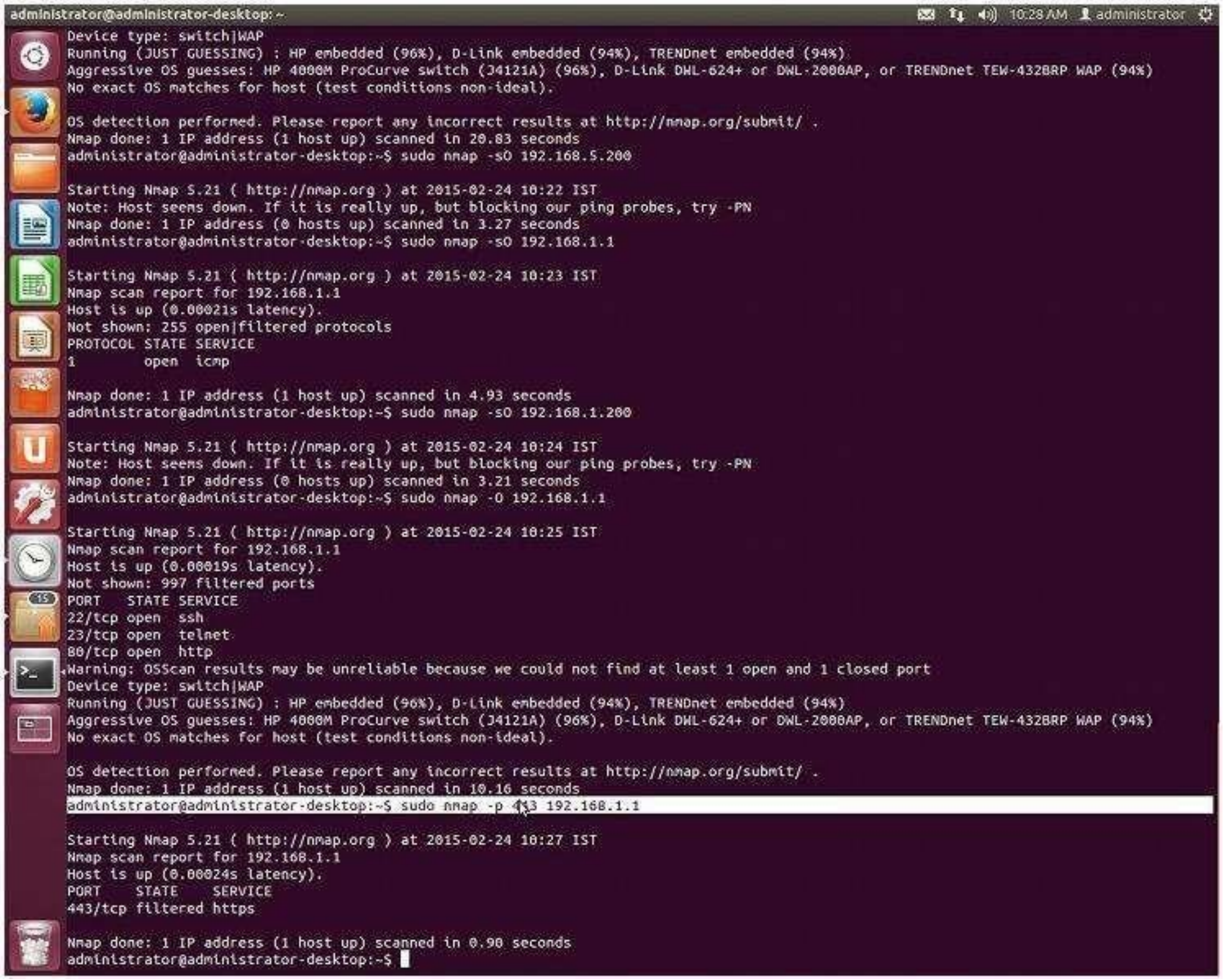


--open (Show only open (or possibly open) ports)

Sometimes you only care about ports you can actually connect to (open ones), and don´t want results cluttered with closed, filtered, and closed|filtered ports.

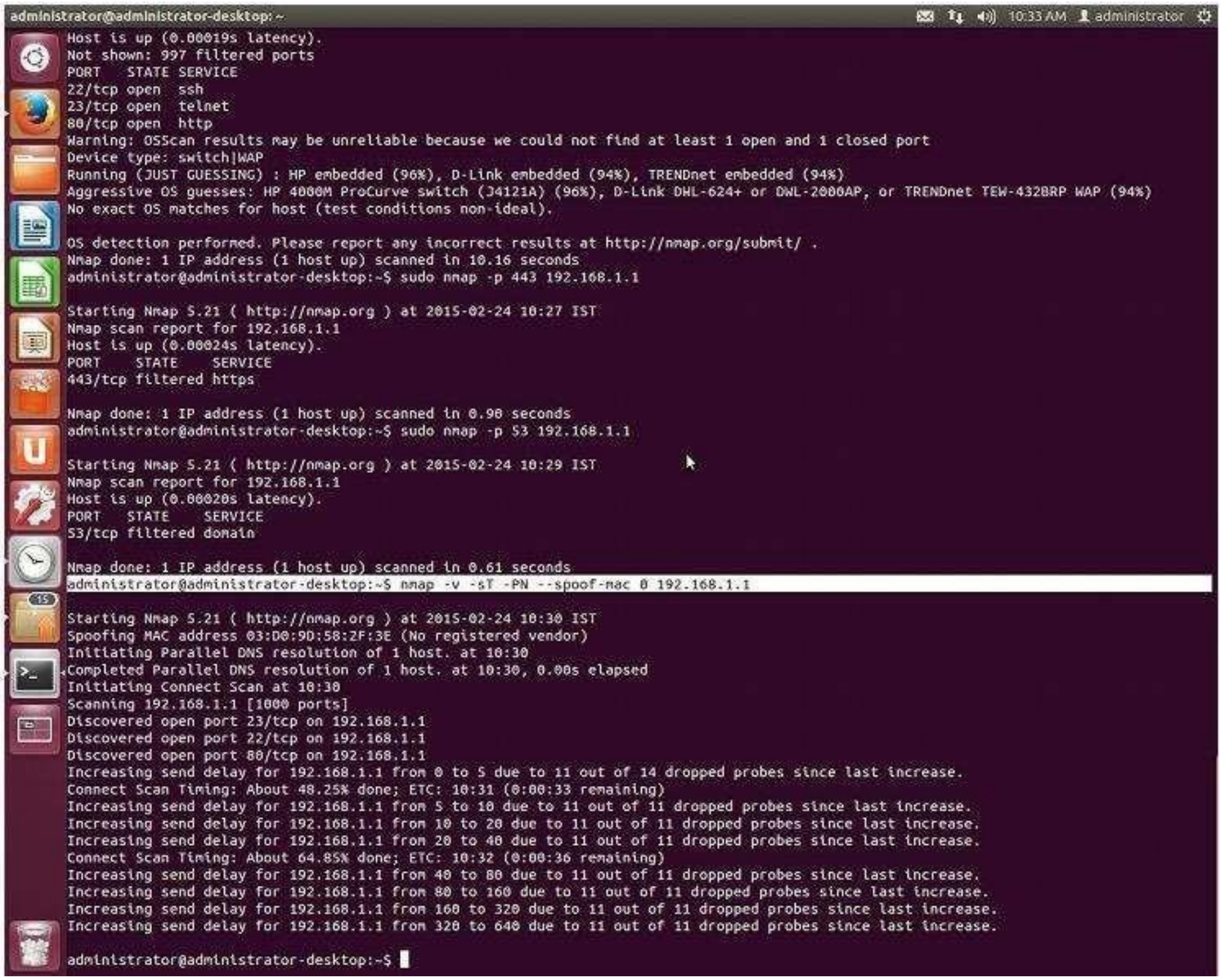
-p port ranges (Only scan specified ports) .

This option specifies which ports you want to scan and overrides the default. Individual port numbers are OK, as are ranges separated by a hyphen (e.g. 1-1023). The beginning and/or end values of a range may be omitted, causing Nmap to use 1 and 65535, respectively.



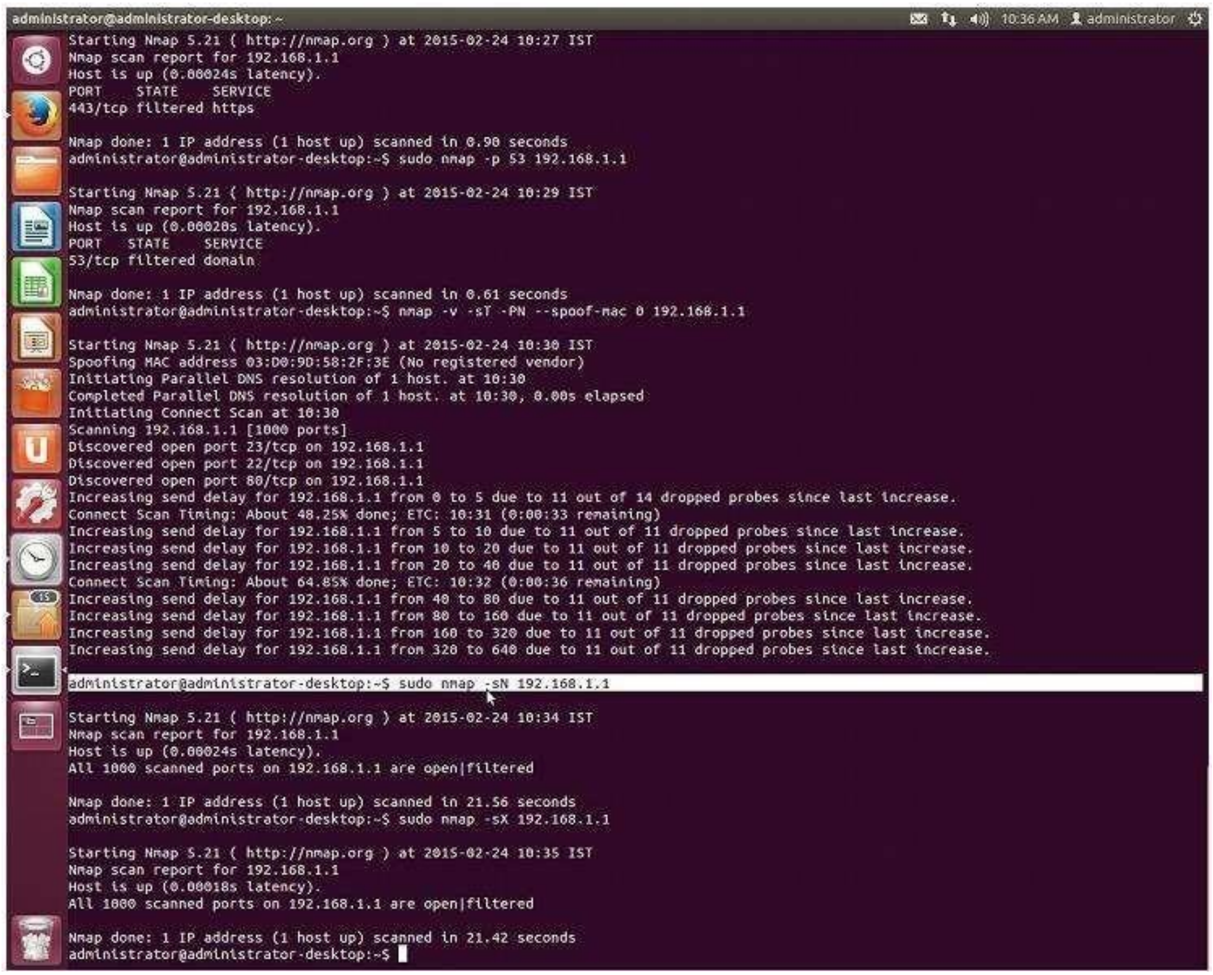
-sT (TCP connect scan) .

TCP connect scan is the default TCP scan type when SYN scan is not an option. This is the case when a user does not have raw packet privileges or is scanning IPv6 networks. Instead of writing raw packets as most other scan types do, Nmap asks the underlying operating system to establish a connection with the target machine and port by issuing the connect system call. Along with spoofing.



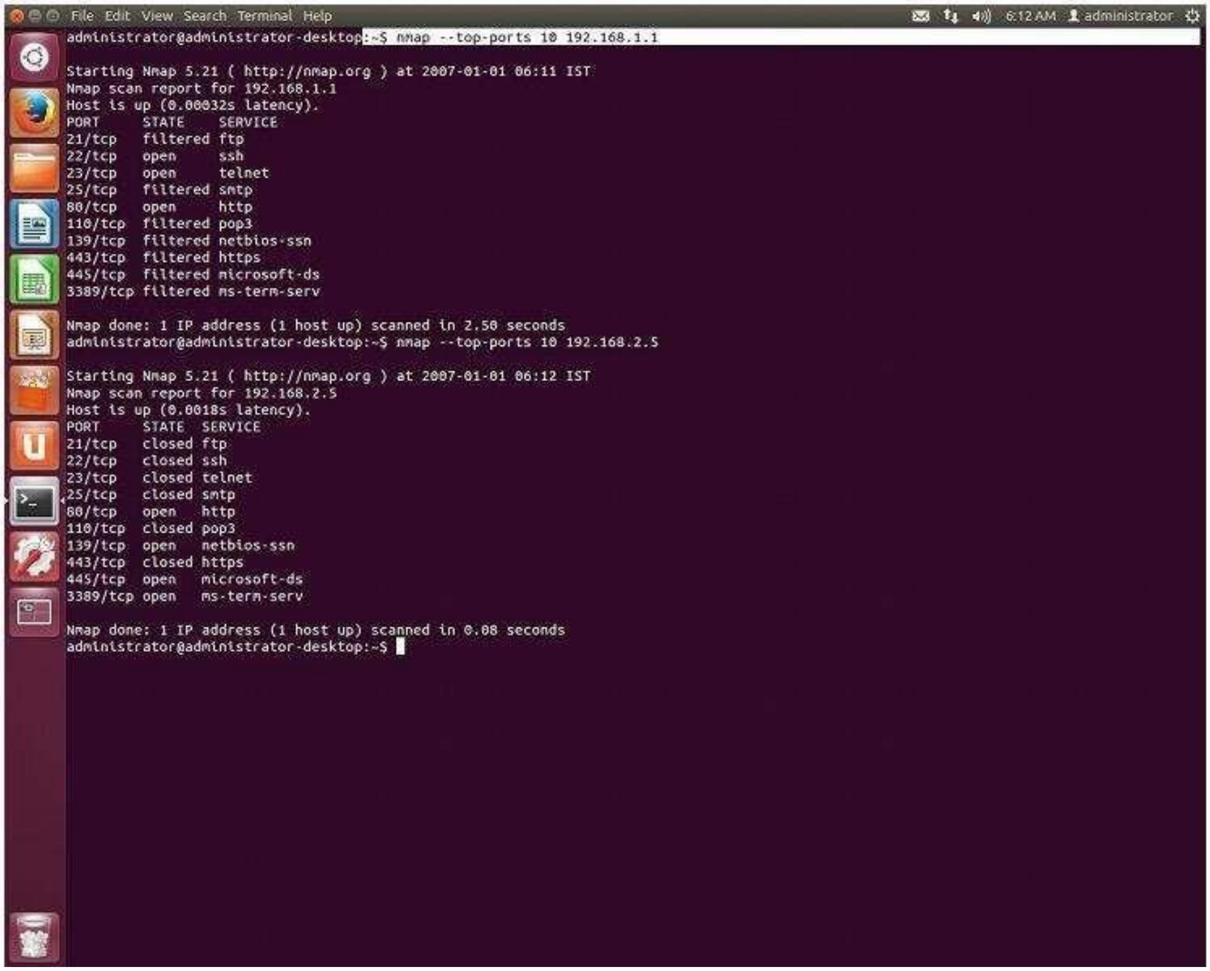
Null scan (-sN):

Does not set any bits (TCP flag header is 0)



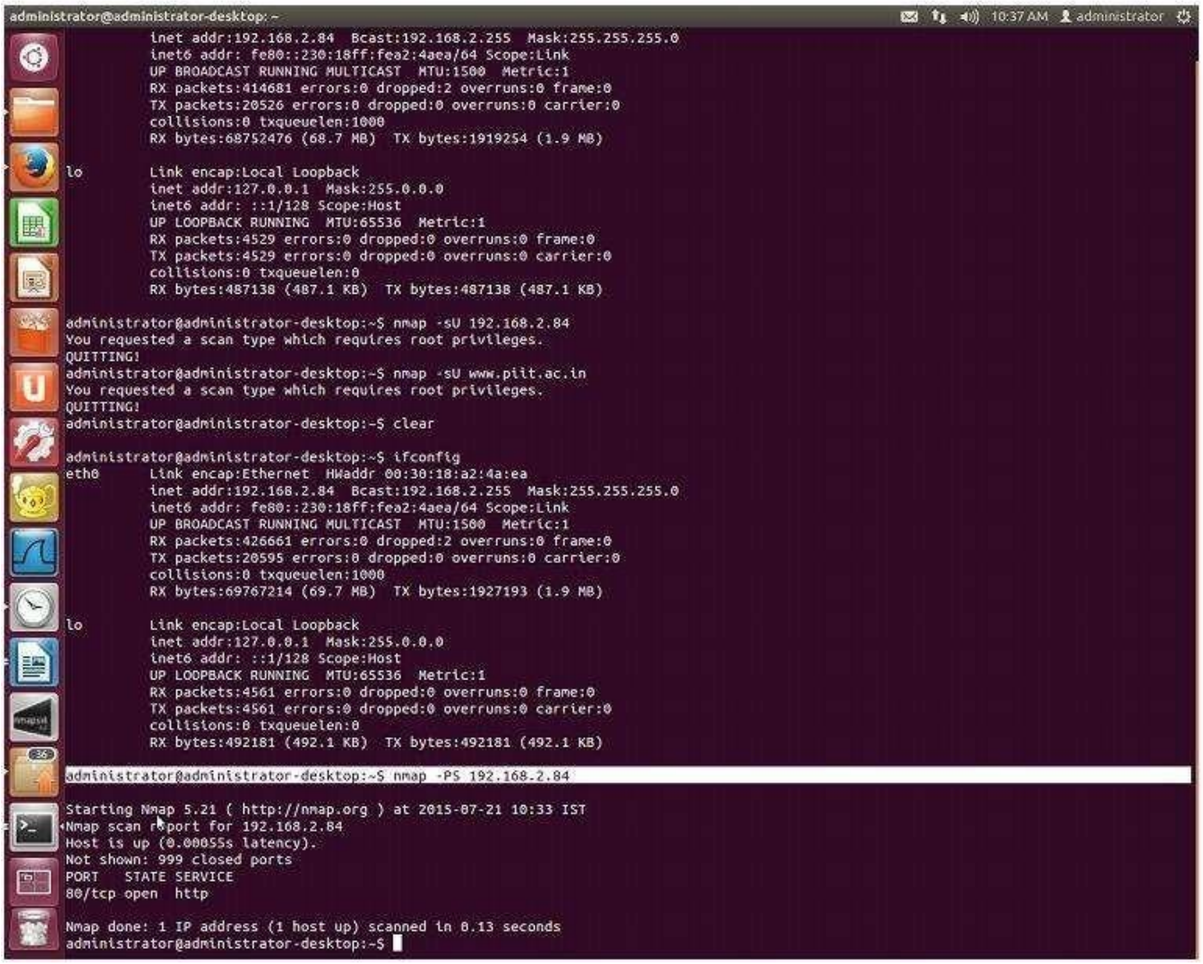
--top-ports <integer of 1 or greater>

Scans the N highest-ratio ports found in nmap-services file.



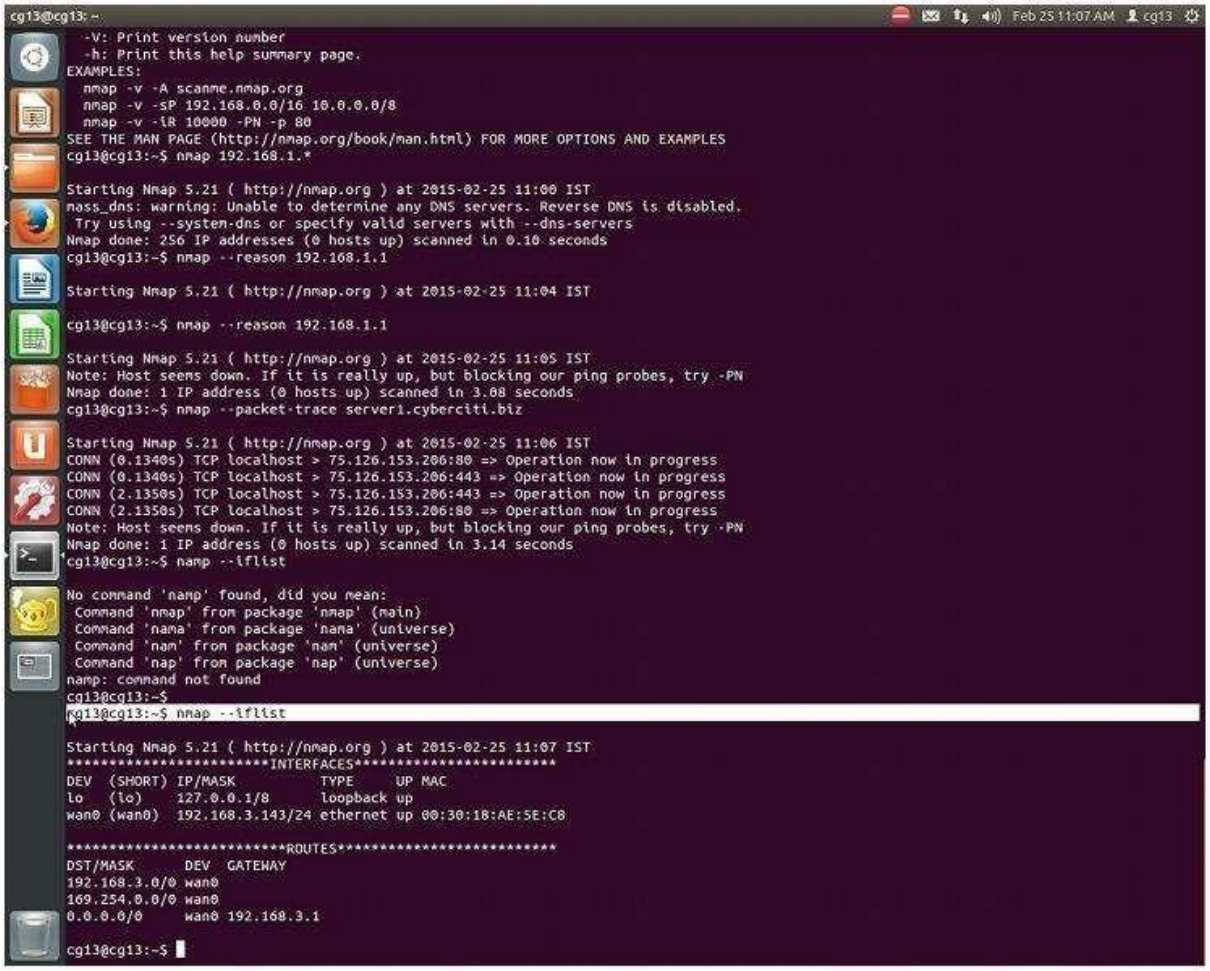
-PS port list (TCP SYN Ping)

This option sends an empty TCP packet with the SYN flag set. The default destination port is 80 (configurable at compile time by changing DEFAULT_TCP_PROBE_PORT_SPEC innmap.h). Alternate ports can be specified as a parameter. The syntax is the same as for the -p except that port type specifiers like T: are not allowed.



nmap –iflist

host interface and route information with nmap by using "–iflist" option.



6. Conclusion:

Network scanning provides a wealth of information about the target network, which is valuable regardless of whether you're trying to attack the network or protect it from attack. While performing a basic scan is a simple matter, the network scanners covered in this experiment provide a wide array of options to tweak your scan to achieve the best results. Nmap is used to detect IP spoofing and port scanning.

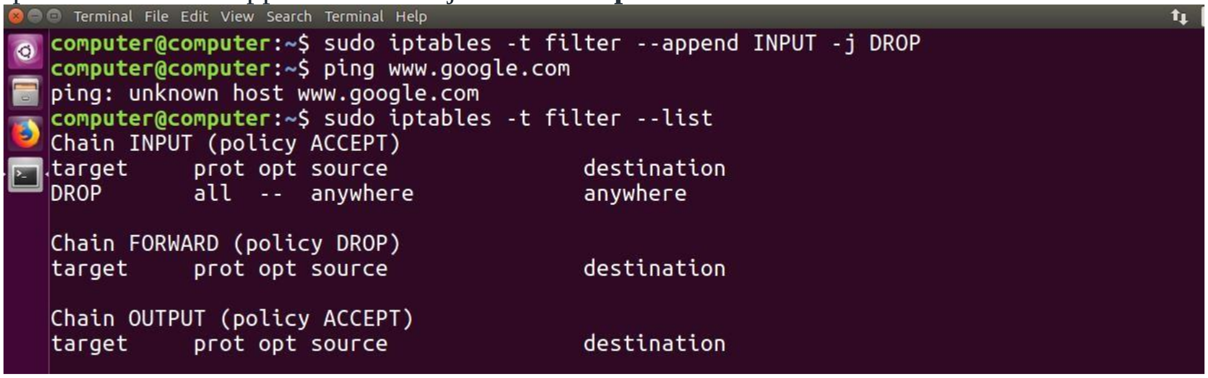
Experiment – 8

1. Aim: Use of iptables in linux
2. Objectives: To study how to create iptables in linux.
3. Theory:
iptables is a command line interface used to set up and maintain tables for the Netfilter firewall for IPv4, included in the Linux kernel. The firewall matches packets with rules defined in these tables and then takes the specified action on a possible match.
Tables is the name for a set of chains.
Chain is a collection of rules.
Rule is condition used to match packet.
Target is action taken when a possible rule matches. Examples of the target are ACCEPT, DROP, QUEUE.
Policy is the default action taken in case of no match with the inbuilt chains and can be ACCEPT or DROP.
Syntax:
iptables --table TABLE -A/-C/-D... CHAIN rule --jump Target
TABLE
There are five possible tables:
filter: Default used table for packet filtering. It includes chains like INPUT, OUTPUT and FORWARD.
nat : Related to Network Address Translation. It includes PREROUTING and POSTROUTING chains.
mangle : For specialised packet alteration. Inbuilt chains include PREROUTING and OUTPUT.
raw : Configures exemptions from connection tracking. Built-in chains are PREROUTING and OUTPUT.
security : Used for Mandatory Access Control
CHAINS
There are few built-in chains that are included in tables. They are:

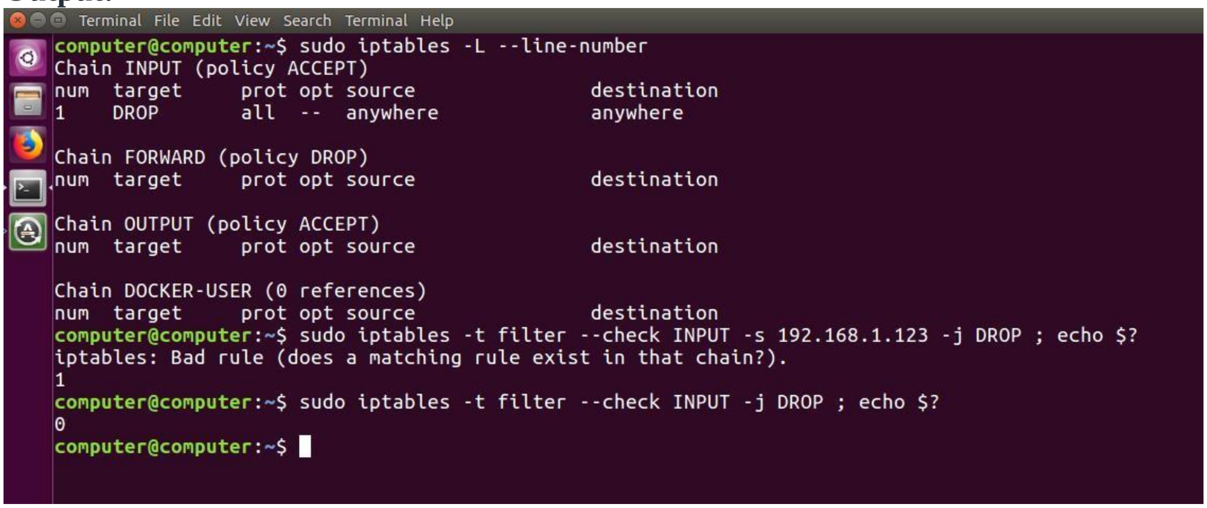
- **INPUT :**set of rules for packets destined to localhost sockets.
- **FORWARD :**for packets routed through the device.
- **OUTPUT :**for locally generated packets, meant to be transmitted outside.
- **PREROUTING :**for modifying packets as they arrive.
- **POSTROUTING :**for modifying packets as they are leaving

OPTIONS

1. **-A, –append :** Append to the chain provided in parameters.
   Syntax: iptables [-t table] --append [chain] [parameters]
   Example: This command drops all the traffic coming on any port.
   iptables -t filter --append INPUT -j DROP **Output:**



2. **-D, –delete :** Delete rule from the specified chain.
   Syntax: iptables [-t table] --delete [chain] [rule_number]
   Example: This command deletes the rule 2 from INPUT chain.
   iptables -t filter --delete INPUT 2
   Output:
1. iptables -t filter --check INPUT -s 192.168.1.123 -j DROP
2. **Output:**

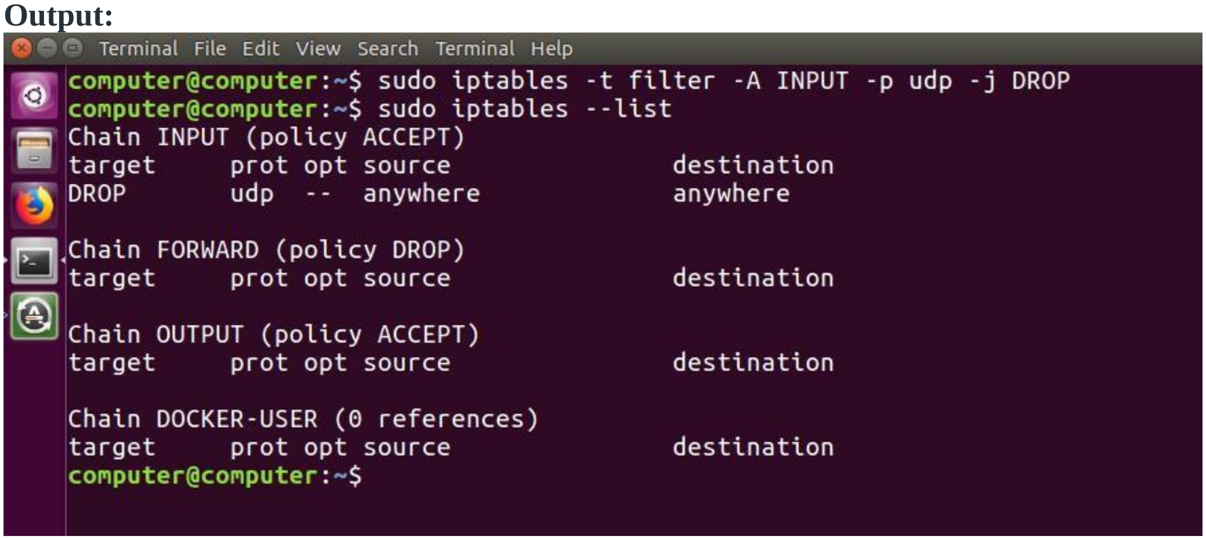


PARAMETERS

The parameters provided with the *iptables* command is used to match the packet and perform the specified action. The common parameters are:

1. **-p, –proto :** is the protocol that the packet follows. Possible values maybe: tcp, udp, icmp, ssh etc.

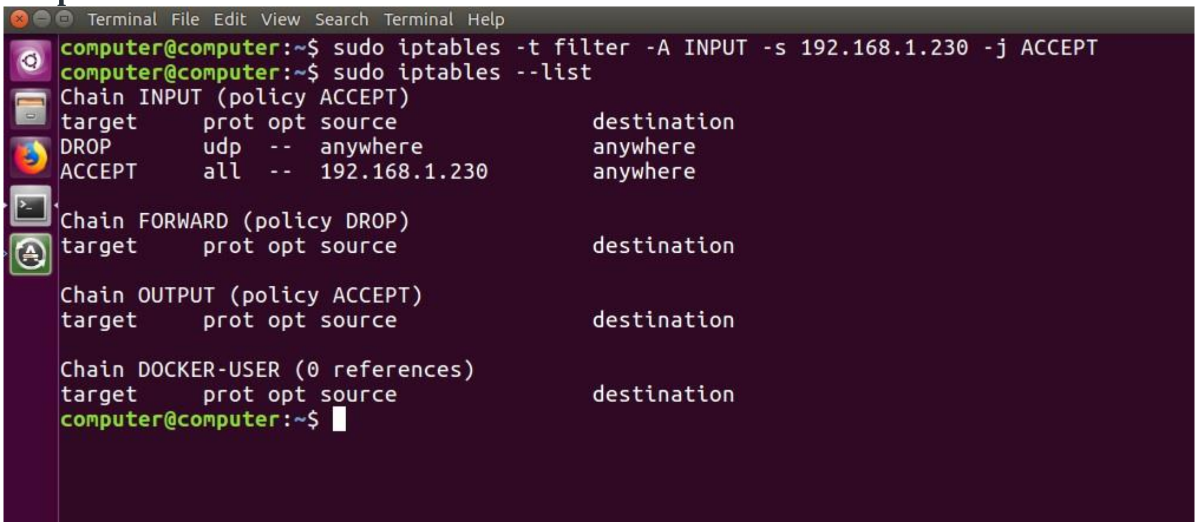
Syntax: iptables [-t table] -A [chain] -p {protocol_name} [target]
Example: This command appends a rule in the INPUT chain to drop all udp packets.
iptables -t filter -A INPUT -p udp -j DROP
Output:



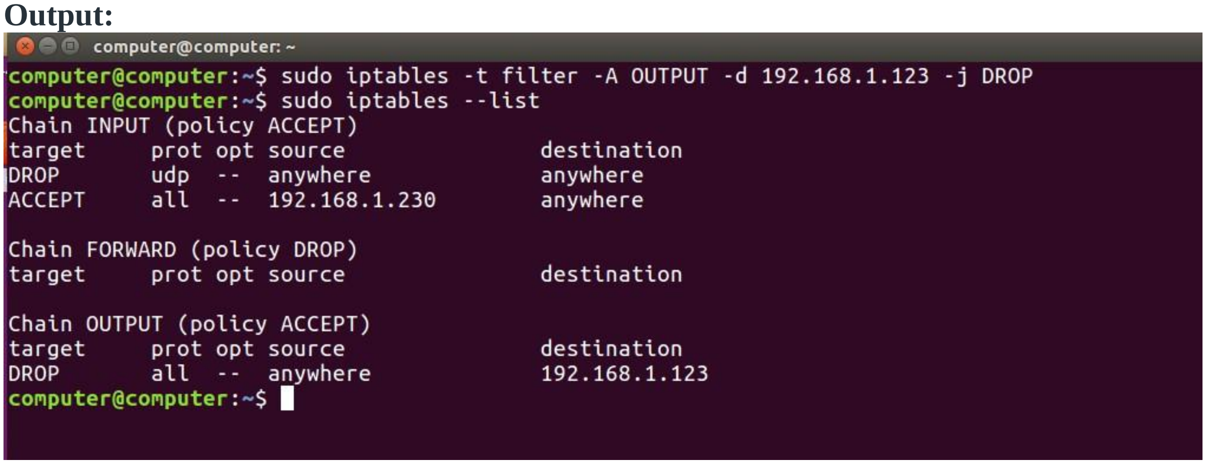
2. **-s, –source:** is used to match with the source address of the packet.

Syntax: iptables [-t table] -A [chain] -s {source_address} [target]

Example: This command appends a rule in the INPUT chain to accept all packets originating from 192.168.1.230.
iptables -t filter -A INPUT -s 192.168.1.230 -j ACCEPT
Output:

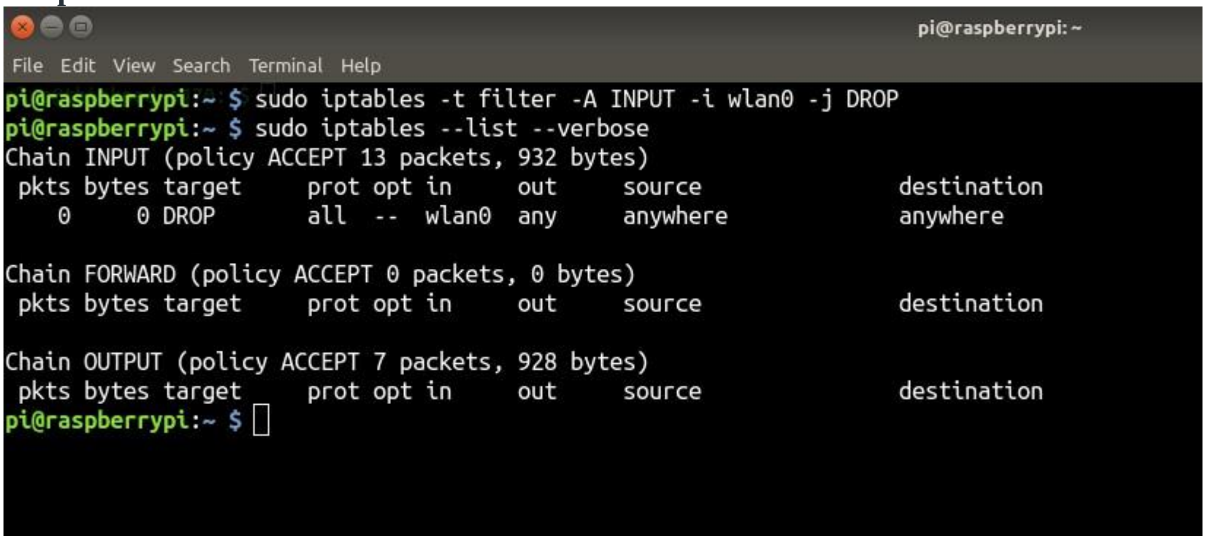


3. **-d, –destination :** is used to match with the destination address of the packet.
Syntax: iptables [-t table] -A [chain] -d {destination_address} [target]
Example: This command appends a rule in the OUTPUT chain to drop all packets destined for 192.168.1.123.
iptables -t filter -A OUTPUT -d 192.168.1.123 -j DROP
Output:



4. **-i, –in-interface :** matches packets with the specified in-interface and takes the action.
Syntax:
iptables [-t table] -A [chain] -i {interface} [target]
Example: This command appends a rule in the INPUT chain to drop all packets destined for wireless interface.
iptables -t filter -A INPUT -i wlan0 -j DROP

Output:



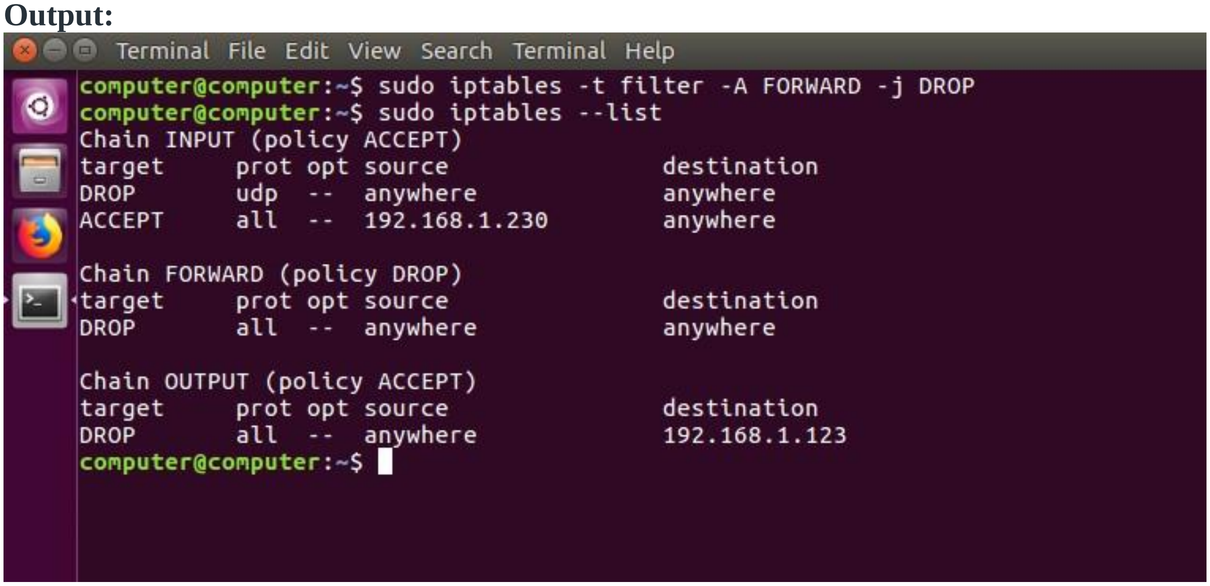
5. **-o, –out-interface :** matches packets with the specified out-interface.
6. **-j, –jump :** this parameter specifies the action to be taken on a match.

Syntax: iptables [-t table] -A [chain] [parameters] -j {target}

Example: This command adds a rule in the FORWARD chain to drop all packets.

iptables -t filter -A FORWARD -j DROP

Output:



- While trying out the commands, you can remove all filtering rules and user created chains.
- sudo iptables --flush
- To save the iptables configuration use:
  sudo iptables-save
- Restoring iptables config can be done with:
  sudo iptables-restore

4. Conclusion:

There are many other firewall utilities and some that may be easier, but iptables is a good learning tool, if only because it exposes some of the underlying netfilter structure and because it is present in so many systems.

Experiment – 9

AIM: To demonstrate Intrusion Detection System (IDS) using Snort software tool.

STEPS ON CONFIGURING AND INTRUSION DETECTION:

1. Download Snort from the Snort.org website. (http://www.snort.org/snortdownloads)

2. Download Rules(https://www.snort.org/snort-rules). You must register to get the rules. (You should download these often)

3. Double click on the .exe to install snort. This will install snort in the “C:\Snort” folder. It is important to have WinPcap(https://www.winpcap.org/install/) installed

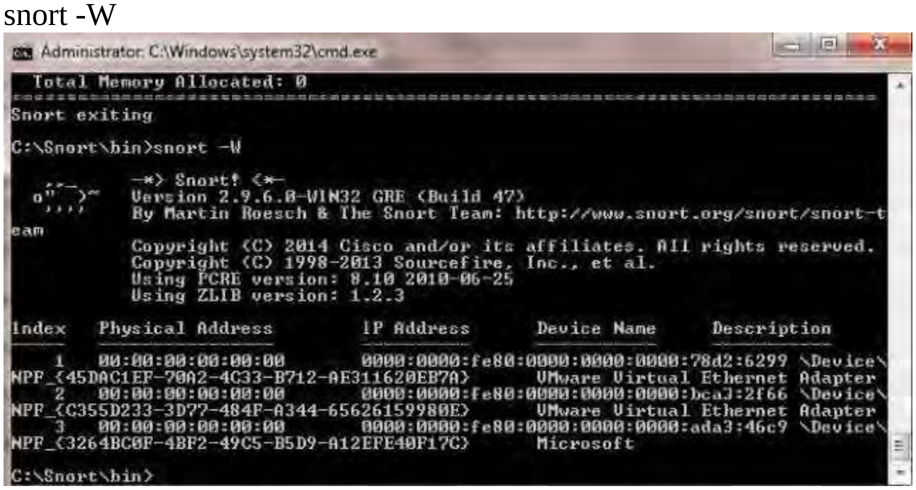
4. Extract the Rules file. You will need WinRAR for the .gz file.

5. Copy all files from the “rules” folder of the extracted folder. Now paste the rules into *“C:\Snort\rules”* folder.

6. Copy “snort.conf” file from the “etc” folder of the extracted folder. You must paste it into “C:\Snort\etc” folder. Overwrite any existing file. Remember if you modify your snort.conf file and download a new file, you must modify it for Snort to work.

7. Open a command prompt (cmd.exe) and navigate to folder “C:\Snort\bin” folder. (at the Prompt, type cd\snort\bin)

8. To start (execute) snort in sniffer mode use following command:
snort -dev -i 3
-i indicates the interface number. You must pick the correct interface number. In my case, it is 3.
-dev is used to run snort to capture packets on your network.
To check the interface list, use following command:
snort -W



Finding an interface

You can tell which interface to use by looking at the Index number and finding Microsoft. As you can see in the above example, the other interfaces are for VMWare. My interface is 3.

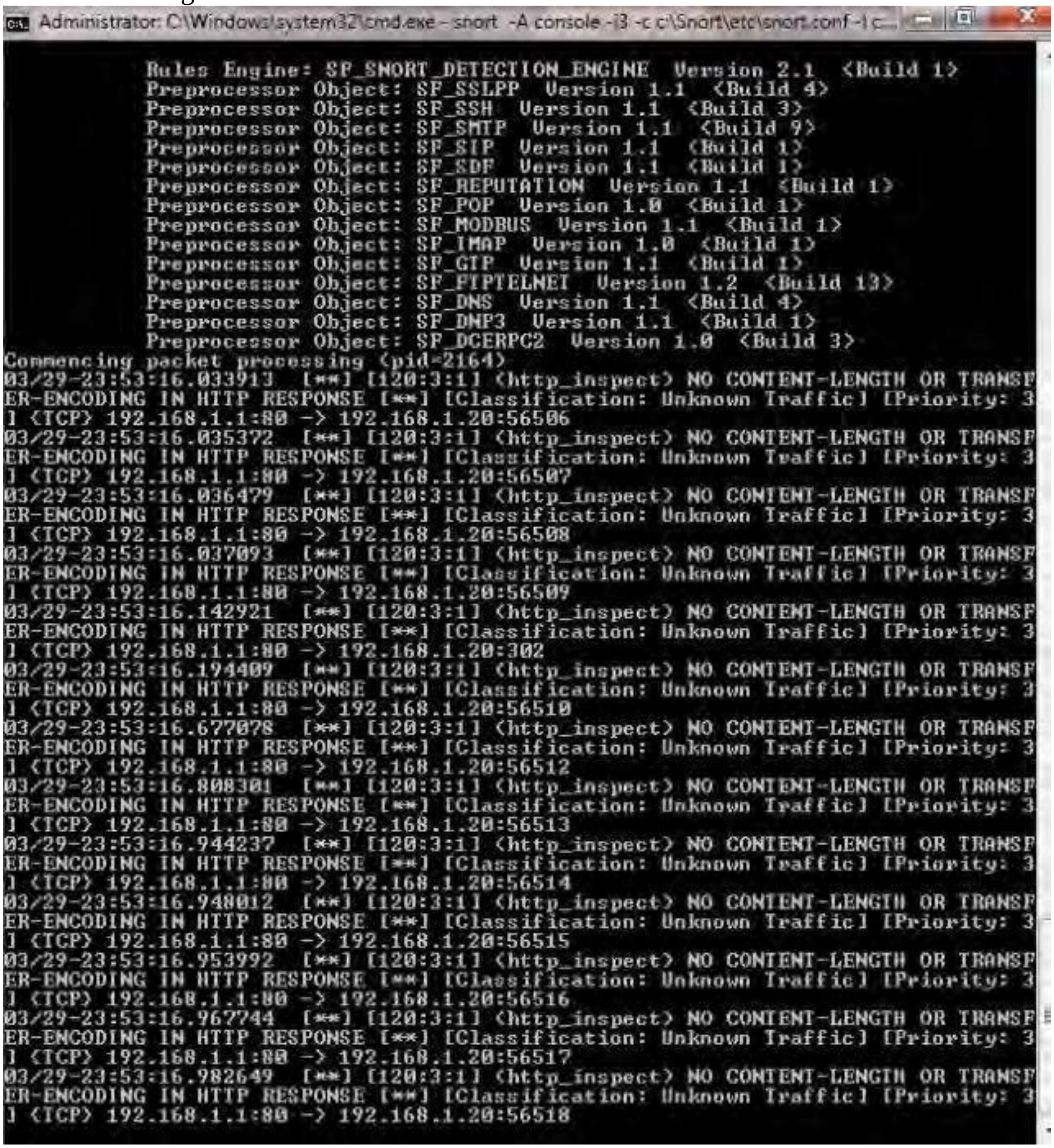
9. To run snort in IDS mode, you will need to configure the file “snort.conf” according to your network environment.

10. To specify the network address that you want to protect in snort.conf file, look for the following line.
var HOME_NET 192.168.1.0/24 (You will normally see any here)

11. You may also want to set the addresses of DNS_SERVERS, if you have some on your network.
Example:

example snort
12. Change the RULE_PATH variable to the path of rules folder.
var RULE_PATH c:\snort\rules
path to rules
13. Change the path of all library files with the name and path on your system. and you must change the path of snort_dynamicpreprocessorvariable.
C:\Snort\lib\snort_dynamiccpreprocessor
You need to do this to all library files in the “C:\Snort\lib” folder. The old path might be: “/usr/local/lib/…”. you will need to replace that path with your system path. Using C:\Snort\lib
14. Change the path of the “dynamicengine” variable value in the “snort.conf” file..
Example:
dynamicengine C:\Snort\lib\snort_dynamicengine\sf_engine.dll
15. Add the paths for “include classification.config” and “include reference.config” files.
include c:\snort\etc\classification.config
include c:\snort\etc\reference.config
16. Remove the comment (#) on the line to allow ICMP rules, if it is commented with a #.
include $RULE_PATH/icmp.rules
17. You can also remove the comment of ICMP-info rules comment, if it is commented.
include $RULE_PATH/icmp-info.rules
18. To add log files to store alerts generated by snort, search for the “output log” test in snort.conf and add the following line:
output alert_fast: snort-alerts.ids
19. Comment (add a #) the whitelist $WHITE_LIST_PATH/white_list.rules and the blacklist
Change the nested_ip inner , \ to nested_ip inner #, \
20. Comment out (#) following lines:
#preprocessor normalize_ip4
#preprocessor normalize_tcp: ips ecn stream
#preprocessor normalize_icmp4
#preprocessor normalize_ip6
#preprocessor normalize_icmp6
21. Save the “snort.conf” file.
22. To start snort in IDS mode, run the following command:
snort -c c:\snort\etc\snort.conf -l c:\snort\log -i 3
(Note: 3 is used for my interface card)
If a log is created, select the appropriate program to open it. You can use WordPard or NotePad++ to read the file.
To generate Log files in ASCII mode, you can use following command while running snort in IDS mode:
snort -A console -i3 -c c:\Snort\etc\snort.conf -l c:\Snort\log -K ascii
23. Scan the computer that is running snort from another computer by using PING or NMap (ZenMap).
After scanning or during the scan you can check the snort-alerts.ids file in the log folder to insure it is

logging properly. You will see IP address folders appear.
Snort monitoring traffic –



RESULT:

Thus the Intrusion Detection System(IDS) has been demonstrated by using the Open Source Snort Intrusion Detection Tool.

VIVA VOICE

1. What is information security, and why is it important in today's digital age?
2. Can you explain the CIA triad in the context of information security?
3. Describe different types of security threats that organizations commonly face.
4. What is the role of encryption in ensuring data security? Can you explain different encryption techniques?
5. How do firewalls contribute to network security? What are the different types of firewalls?
6. Discuss the importance of access control in information security. What are the different access control models?
7. Explain the concept of authentication and its significance in information security.
8. What are the main differences between symmetric and asymmetric encryption algorithms?
9. Describe common social engineering techniques used to breach security and how to prevent them.
10. How does a Distributed Denial of Service (DDoS) attack work, and what measures can be taken to mitigate it?
11. What are the best practices for securing wireless networks?
12. Explain the concept of digital signatures and their role in ensuring data integrity and authenticity.
13. What is a vulnerability assessment, and how does it differ from penetration testing?
14. Discuss the importance of security policies and procedures in an organization.
15. How do encryption protocols like SSL/TLS contribute to securing communication over the internet?
16. Can you explain the concept of risk management in information security? What are the steps involved?
17. Describe the main principles of secure software development.
18. How does biometric authentication work, and what are its advantages and disadvantages?
19. What are the ethical considerations in information security, especially regarding privacy and data protection?
20. Discuss the emerging trends and challenges in information security, such as IoT security, AI-based threats, and quantum computing implications.