Experiment 1

Date of Performance : **17-03-2022** **Date of Submission:** **19-03-2022**

SAP Id: 60004200059 **Name : Vaishnavi Sawant**

Div: A **Batch : A2**

Aim of Experiment

Design and Implement Encryption and Decryption Algorithm for

a. Caesar cipher cryptographic algorithm by considering letter [A..Z] and digits [0..9]. Create two functions Encrypt() and Decrypt(). Apply Brute Force Attack to reveal secret. Create Function BruteForce(). Demonstrate the use of these functions on any paragraph.
b. Affine Cipher. Your Program Must Input Image in Gray Scale. Choose keys according to Gray Scale Intensity level. Create two functions Encrypt() and Decrypt(). Make sure to have Multiplicative Inverse Exists for one of the Key in selected Key pair of Affine Cipher. (CO1)

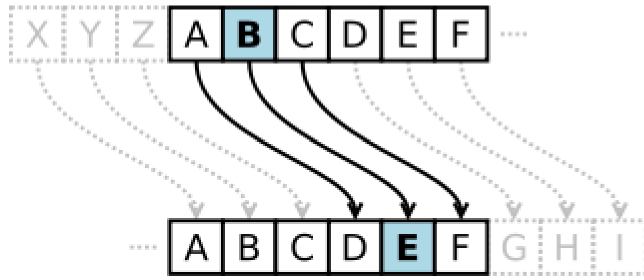
Theory:

Caesar Cipher:

In cryptography, a Caesar cipher, also known as a Caesar's cipher, the shift cipher, Caesar's Code or Caesar Shift, is one of the simplest and most widely-known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions further down the alphabet . For example, with a shift of 3, A would be replaced by D, B would become E, and so on. The method is named after Julius Caesar, who used it to communicate with his generals.

The encryption step performed by a Caesar cipher is often incorporated as part of more complex schemes, such as the Vigenère cipher, and still has modern application in the ROT13 system. As with all single alphabet substitution ciphers, the Caesar cipher is easily broken and in practice offers essentially no communication security.

The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1,…, Z = 25. Encryption of a letter by a shift n can be described mathematically as.



Affine Cipher:

Affine cipher is a monoalphabetical symmetrical substitution cipher, which eliminates the biggest drawback of the Caesar cipher – very easy cryptanalysis stemming from the low number of possible transformations. In Affine Cipher each letter in an alphabet is mapped to its numeric equivalent, encrypted using a simple mathematical function, and converted back to a letter. The formula used means that each letter encrypts to one other letter, and back again, meaning the cipher is essentially a standard substitution cipher with a rule governing which letter goes to which. As such, it has the weaknesses of all substitution ciphers. Each letter is enciphered with the function (ax + b) mod 26, where b is the magnitude of the shift.

Encryption

It uses modular arithmetic to transform the integer that each plaintext letter corresponds to into another integer that correspond to a ciphertext letter. The encryption function for a single letter is
E (x) = (a x + b) mod m
modulus m: size of the alphabet
a and b: key of the cipher.
a must be chosen such that a and m are coprime.

Decryption

In deciphering the ciphertext, we must perform the opposite (or inverse) functions on the ciphertext to retrieve the plaintext. Once again, the first step is to convert each of the ciphertext letters into their integer values. The decryption function is
D (x) = a^-1 (x - b) mod m
a^-1 : modular multiplicative inverse of a modulo m. i.e., it satisfies the equation
1 = a a^-1 mod m .

Program

A)

```
def Encrypt(plainText,key):
    ciphertext = ""
    for i in range(0,len(plainText)):
        ch = plainText[i]
        if ch==' ':
            ciphertext+=" "
        elif ch.isnumeric():
            asci_val = (ord(ch) - 48 +key)%10
            ciphertext+= chr(48 + asci_val)
        else:
            asci_val = (ord(ch)-65+key)%26
            ciphertext+=chr(65+asci_val)

    return ciphertext

def Decrypt(cipherText, key):
    plainText = ""
    for i in range(0, len(cipherText)):
        ch = cipherText[i]
        if ch == " ":
            plainText += " "
        elif ch.isnumeric():
            if key > 9:
                key %= 10
            ascii_val = (ord(ch) - 48 - key)
            if ascii_val < 0:
                ascii_val =  abs(ascii_val) % 10
                plainText += chr(57 - ascii_val + 1)
            else:
                ascii_val =  abs(ascii_val) % 10
                plainText += chr(48 + ascii_val)
        else:
            if key > 25:
                key %= 26
            ascii_val = (ord(ch) - 65 - key)
            if ascii_val < 0:
                ascii_val =  abs(ascii_val) % 26
                plainText += chr(90 - ascii_val + 1)
            else:
                ascii_val =  abs(ascii_val) % 26
                plainText += chr(65 + ascii_val)
    print("Decrypted Text: ", plainText)
```

```
def BruteForce(cipherText, key):
    if key == 26:
        return
    else:
        print("Brute Force: Key=", key)
        Decrypt(cipherText, key)
        print("########################################")
        BruteForce(cipherText, key+1)

ckey=int(input("Enter key for encryption:"))
pt=input("Enter plain text for encryption:")
cipher = Encrypt(pt,ckey)
print("Cipher Text:",cipher)
Decrypt(cipher, ckey)
BruteForce(cipher, 0)
```

Output

```
Enter key for encryption:8
Enter plain text for encryption:A TRULY HAPPY PERSON IS ONE WHO CAN ENJOY THE SCENERY WHILE ON A DETOUR
Cipher Text: I BZCTG PIXXG XMZAWV QA WVM EPW KIV MVRWG BPM AKMVMZG EPQTM WV I LMBWCZ
Decrypted Text:  A TRULY HAPPY PERSON IS ONE WHO CAN ENJOY THE SCENERY WHILE ON A DETOUR
Brute Force: Key= 0
Decrypted Text:  I BZCTG PIXXG XMZAWV QA WVM EPW KIV MVRWG BPM AKMVMZG EPQTM WV I LMBWCZ
#############################################
Brute Force: Key= 1
Decrypted Text:  H AYBSF OHWWF WLYZVU PZ VUL DOV JHU LUQVF AOL ZJLULYF DOPSL VU H KLAVBY
#############################################
Brute Force: Key= 2
Decrypted Text:  G ZXARE NGVVE VKXYUT OY UTK CNU IGT KTPUE ZNK YIKTKXE CNORK UT G JKZUAX
#############################################
Brute Force: Key= 3
Decrypted Text:  F YWZQD MFUUD UJWXTS NX TSJ BMT HFS JSOTD YMJ XHJSJWD BMNQJ TS F IJYTZW
#############################################
Brute Force: Key= 4
Decrypted Text:  E XVYPC LETTC TIVWSR MW SRI ALS GER IRNSC XLI WGIRIVC ALMPI SR E HIXSYV
#############################################
Brute Force: Key= 5
Decrypted Text:  D WUXOB KDSSB SHUVRQ LV RQH ZKR FDQ HQMRB WKH VFHQHUB ZKLOH RQ D GHWRXU
#############################################
Brute Force: Key= 6
Decrypted Text:  C VTWNA JCRRA RGTUQP KU QPG YJQ ECP GPLQA VJG UEGPGTA YJKNG QP C FGVQWT
#############################################
Brute Force: Key= 7
Decrypted Text:  B USVMZ IBQQZ QFSTPO JT POF XIP DBO FOKPZ UIF TDFOFSZ XIJMF PO B EFUPVS
#############################################
Brute Force: Key= 8
Decrypted Text:  A TRULY HAPPY PERSON IS ONE WHO CAN ENJOY THE SCENERY WHILE ON A DETOUR
#############################################
Brute Force: Key= 9
Decrypted Text:  Z SQTKX GZOOX ODQRNM HR NMD VGN BZM DMINX SGD RBDMDQX VGHKD NM Z CDSNTQ
#############################################
Brute Force: Key= 10
Decrypted Text:  Y RPSJW FYNNW NCPQML GQ MLC UFM AYL CLHMW RFC QACLCPW UFGJC ML Y BCRMSP
#############################################
Brute Force: Key= 11
Decrypted Text:  X QORIV EXMMV MBOPLK FP LKB TEL ZXK BKGLV QEB PZBKBOV TEFIB LK X ABQLRO
#############################################
Brute Force: Key= 12
Decrypted Text:  W PNQHU DWLLU LANOKJ EO KJA SDK YWJ AJFKU PDA OYAJANU SDEHA KJ W ZAPKQN
#############################################
Brute Force: Key= 13
Decrypted Text:  V OMPGT CVKKT KZMNJI DN JIZ RCJ XVI ZIEJT OCZ NXZIZMT RCDGZ JI V YZOJPM
#############################################
Brute Force: Key= 14
Decrypted Text:  U NLOFS BUJJS JYLMIH CM IHY QBI WUH YHDIS NBY MWYHYLS QBCFY IH U XYNIOL
#############################################
```

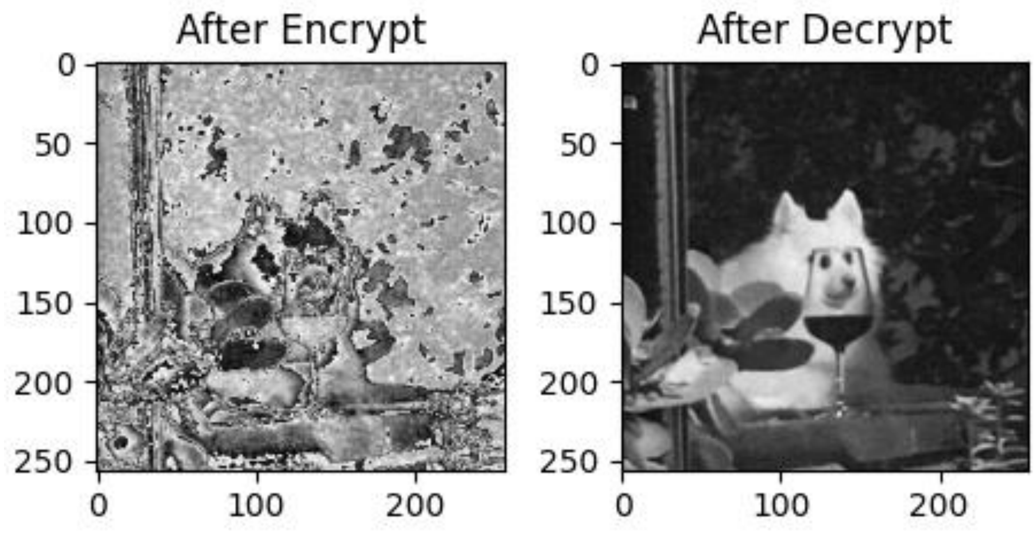
```
Brute Force: Key= 15
Decrypted Text:  T MKNER ATIIR IXKLHG BL HGX PAH VTG XGCHR MAX LVXGXKR PABEX HG T WXMHNK
##############################################
Brute Force: Key= 16
Decrypted Text:  S LJMDQ ZSHHQ HWJKGF AK GFW OZG USF WFBGQ LZW KUWFWJQ OZADW GF S VWLGMJ
##############################################
Brute Force: Key= 17
Decrypted Text:  R KILCP YRGGP GVIJFE ZJ FEV NYF TRE VEAFP KYV JTVEVIP NYZCV FE R UVKFLI
##############################################
Brute Force: Key= 18
Decrypted Text:  Q JHKBO XQFFO FUHIED YI EDU MXE SQD UDZEO JXU ISUDUHO MXYBU ED Q TUJEKH
##############################################
Brute Force: Key= 19
Decrypted Text:  P IGJAN WPEEN ETGHDC XH DCT LWD RPC TCYDN IWT HRTCTGN LWXAT DC P STIDJG
##############################################
Brute Force: Key= 20
Decrypted Text:  O HFIZM VODDM DSFGCB WG CBS KVC QOB SBXCM HVS GQSBSFM KVWZS CB O RSHCIF
##############################################
Brute Force: Key= 21
Decrypted Text:  N GEHYL UNCCL CREFBA VF BAR JUB PNA RAWBL GUR FPRAREL JUVYR BA N QRGBHE
##############################################
Brute Force: Key= 22
Decrypted Text:  M FDGXK TMBBK BQDEAZ UE AZQ ITA OMZ QZVAK FTQ EOQZQDK ITUXQ AZ M PQFAGD
##############################################
Brute Force: Key= 23
Decrypted Text:  L ECFWJ SLAAJ APCDZY TD ZYP HSZ NLY PYUZJ ESP DNPYPCJ HSTWP ZY L OPEZFC
##############################################
Brute Force: Key= 24
Decrypted Text:  K DBEVI RKZZI ZOBCYX SC YXO GRY MKX OXTYI DRO CMOXOBI GRSVO YX K NODYEB
##############################################
Brute Force: Key= 25
Decrypted Text:  J CADUH QJYYH YNABXW RB XWN FQX LJW NWSXH CQN BLNWNAH FQRUN XW J MNCXDA
##############################################
```

Program

B)

```
import numpy as np
import PIL
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = [7.00, 3.50]
plt.rcParams["figure.autolayout"] = True

def imageImport(file):
    img = PIL.Image.open(file)
    gray_img = img.convert("L")
    print(gray_img.size)
    imgArr = np.array(gray_img)
    return imgArr

def mul_Inverse(a,b):
    ri = [b,a]
    qi=[0,0]
    xi=[1,0]
    yi=[0,1]
    i=2
    while True:
        ri.append(ri[i-2]%ri[i-1])
        qi.append(ri[i-2]//ri[i-1])
        xi.append(xi[i-2]-qi[i]*xi[i-1])
        yi.append(yi[i-2]-qi[i]*yi[i-1])
        if ri[i]==1:
            break
        i+=1

    XI=(b+xi[-1])if(xi[-1]<0)else(xi[-1])
    YI=(b+yi[-1])if(yi[-1]<0)else(yi[-1])
    return YI

def Encrypt(x,a,b,size):
    Ex = (a*x+b)%size
    return Ex

def Decrypt(Ex,Yi,b,size):
    PT = (Yi*(Ex-b))%size
    return PT

imgArr = imageImport("One.jpg")
pr_Ex = Encrypt(imgArr,5,8,256)
Yi = mul_Inverse(5,256)
PT = Decrypt(pr_Ex,Yi,8,256)

plt.subplot(1, 3, 1)
plt.title("Before Encrypt")
plt.imshow(imgArr, cmap='gray')

plt.subplot(1, 3, 2)
plt.title("After Encrypt")
plt.imshow(pr_Ex, cmap='gray')

plt.subplot(1, 3, 3)
plt.title("After Decrypt")
plt.imshow(PT, cmap='gray')

plt.show()
```

Output



Conclusion:

Thus through this experiment we understood the conceptual, mathematical and practical aspect of Caesar Cipher and Affine Cipher Techniques. We implemented Caesar for alphabets as well as numbers and performed affine cipher on gray scale image using python.

Experiment 2

Date of Performance : 24-03-2022 **Date of Submission: 25-03-2022**

SAP Id: 60004200059 **Name : Vaishnavi Sawant**

Div: A **Batch : A2**

Aim of Experiment

Design and Implement Encryption and Decryption algorithm using Simple Columnar Transposition cipher technique. (CO1)

Theory:

Simple Columnar Transposition cipher:

In cryptography, a transposition cipher is a method of encryption by which the positions held by units of plaintext (which are commonly characters or groups of characters) are shifted according to a regular system, so that the ciphertext constitutes a permutation of the plaintext. That is, the order of the units is changed (the plaintext is reordered).
In a columnar transposition cipher, the message is written in a grid of equal length rows, and then read out column by column. The columns are chosen in a scrambled order, decided by the encryption key. Since transposition ciphers doesn't affect the letter frequencies, it can be detected through frequency analysis

Algorithm

The name of the cipher comes after the operations on a matrix, that are performed during both, encryption and decryption. The number of columns of the matrix is determined by the secret key.
The secret key is usually a word (or just a sequence of letters). It has to be converted into a sequence of numbers. The numbers are defined by an alphabetical order of the letters in the keyword. The letter which is first in the alphabet will be the number 1, the second letter in the alphabetical order will be 2, and so on.
If there are multiple identical letters in the keyword, each next occurrence of the same letter should be converted into a number that is equal to the number for the previous occurrence increased by one.
For example, the keyword:
SWINDON
would produce the following sequence of numbers:
6723154
We can see, that we converted the letters N into the numbers 3 and 4.
To encrypt a message, all the letters should be entered into the matrix, row by row, from left to right. The size of the matrix depends on the length of the message. The only known

dimension is width, which is determined by the length of the secret keyword (which is the same as the length of the corresponding sequence of numbers), and known to both sides of the communication.

If, after entering the whole message, there are some empty cells in the bottom row of the matrix, one of two approaches can be taken:

1. The cells may be left empty, and just ignored during all further operations (this is so called an irregular columnar transposition cipher).
2. The sender may enter there some rare letters, and treat them as a part of the plaintext. After decryption, the receiver should be able to determine, that the letters have no sense, and that they should be ignored (in this case, the cipher is called a regular columnar transposition cipher).

Next, the letters should be read off in a specific way, and write down to form the ciphertext. The order of reading the letters is determined by the sequence of numbers, produced from the keyword. They should be read column by column, from top to bottom, starting from the column, which position is the same as the position of the number 1 in the key sequence. The next column to read off is determined by the number 2 in the key sequence, and so on, until all the columns are read off. To make this step easier, it is recommended to write the sequence numbers above the corresponding columns.

As an example, let's encrypt a message A Midsummer Night's Dream, which is a comedy written by Shakespeare. We will use the secret key mentioned above. The number sequence derived from this keyword is 6723154, so the matrix created for the encryption will have seven columns.

After removing all non-letter characters, and changing the letters to upper case, the message should be entered into the table:

6	7	2	3	1	5	4
A	M	I	D	S	U	M
M	E	R	N	I	G	H
T	S	D	R	E	A	M

Above the message, there are numbers derived from the keyword. These numbers determine the order, in which the columns should be read (top to bottom), and appended to the produced ciphertext. In our example, the first column will be SIE, the second will be IRD, and so on.

The produced ciphertext is:

SIE IRD DNR MHM UGA AMT MES

Finally, after removing the spaces, which were added to indicate separate columns, we receive the encrypted message:

SIEIRDDNRMHMUGAAMTMES

To decrypt a received ciphertext, the receiver has to perform the following steps:

1. Knowing the secret keyword, and the length of the received message, the table of the same size, as the one used for encryption, should be created.
2. The ciphertext should be entered into columns, from the leftmost columns to the rightmost column, from top to bottom.
3. The columns should be rearranged, and put into the order defined by the keyword.
4. The decrypted message should be read out, row by row, starting from the top row, and from left to right.

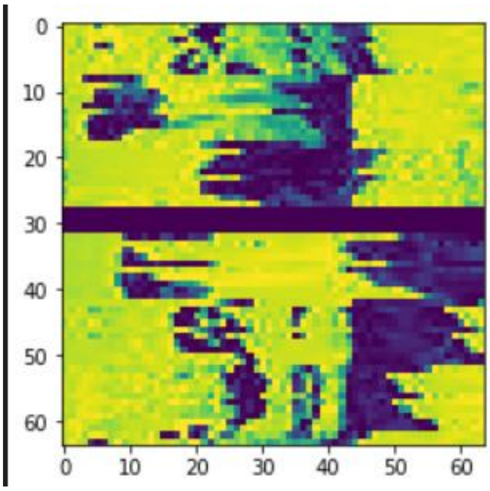
Program:

```
import numpy as np
import PIL
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = [7.00, 3.50]
plt.rcParams["figure.autolayout"] = True

def imageImport(file):
    img = PIL.Image.open(file)
    gray_img = img.convert("L")
    print(gray_img.size)
    imgArr = np.array(gray_img)
    return imgArr

matrix=imageImport('images_sm.jpg')

def encryption(image,key):
  inputImageShape=image.shape
  inputrows=inputImageShape[0]
  n=len(key)
  sortedKey=sorted(key)
  divisible=inputImageShape[1]//len(sortedKey)

  numsRows=inputImageShape[0]
  numCols=divisible*len(sortedKey)+inputImageShape[1]%len(sortedKey)
  matrix1=np.zeros((numsRows,numCols))

  hashmap = { i : sortedKey[i] for i in range(0, len(sortedKey) ) }

  hashmap = {value:key for key, value in hashmap.items()}

  print(hashmap)

  transpose=matrix.T
  transpose1=matrix1.T
  for i in range(0,numCols-inputImageShape[1]%len(sortedKey)):
      currentItem=key[i%len(key)]
      currentIndex=hashmap[currentItem]
      a = i//len(key)
      b = len(key)*a
      c = b + currentIndex
      transpose1[c]=transpose[i]
  final = transpose1.T
  plt.imshow(final)
  flattening=final.flatten()
  rolling=np.roll(flattening,numCols//2)
  reshaping=rolling.reshape(numsRows,numCols)
  transposing = reshaping.T
```

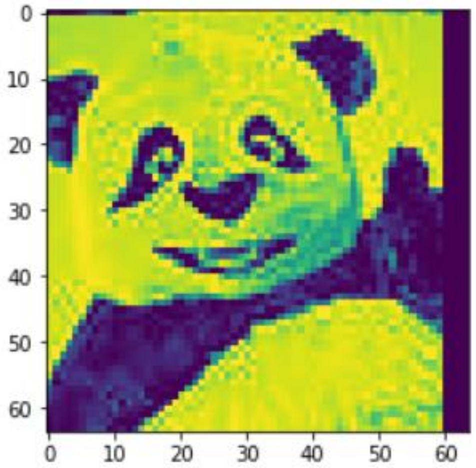
```
  plt.imshow(transposing)
  return transposing

Cipherimage = encryption(matrix,'TRAMPOLINE')

def decryption(matrix,key):

  decrypttranspose = matrix.T
  inputImageShape=decrypttranspose.shape
  inputrows=inputImageShape[0]
  n=len(key)
  sortedKey=sorted(key)
  divisible=inputImageShape[1]//len(sortedKey)

  numsRows=inputImageShape[0]
  numCols=divisible*len(sortedKey)+inputImageShape[1]%len(sortedKey)
  matrix1=np.zeros((numsRows,numCols))

  hashmap = { i : sortedKey[i] for i in range(0, len(sortedKey) ) }

  hashmap = {value:key for key, value in hashmap.items()}

  reflattening=decrypttranspose.flatten()
  rerolling=np.roll(reflattening,numCols//2)
  retranspose=rerolling.reshape(numsRows,numCols)
  plt.imshow(retranspose)

  finaldecrypt=np.zeros((numsRows,numCols))

  retranspose1=retranspose.T
  for i in range(0,numCols-inputImageShape[1]%len(sortedKey)):
      currentItem=key[i%len(key)]
      currentIndex=hashmap[currentItem]
      a = i//len(key)
      b = len(key)*a
      c = b + currentIndex
      finaldecrypt[i]=retranspose1[c]
  finaldecrypt = finaldecrypt.T

  plt.imshow(finaldecrypt)

decryption(Cipherimage,'TRAMPOLINE')
```

Output:

After Encryption:



After Decryption



Conclusion:

Hence we learnt the working of Columnar Transposition Cipher how it rearranges the plaintext letters, based on a matrix filled with letters in the order determined by the secret keyword. We applied this technique to an image where the pixels of the image are shuffled in order to procure an encrypted image.

Experiment 3

Date of Performance : **31-03-2022** **Date of Submission:** **03-04-2022**

SAP Id: 60004200059 **Name : Vaishnavi Sawant**

Div: A **Batch : A2**

Aim of Experiment

Implement Simplified Data Encryption Standard (S-DES)

Theory:

Simplified Data Encryption Standard (S-DES) is a simple version of the DES Algorithm. It is a symmetric key cipher i.e. they use the same key for both encryption and decryption.
Features:

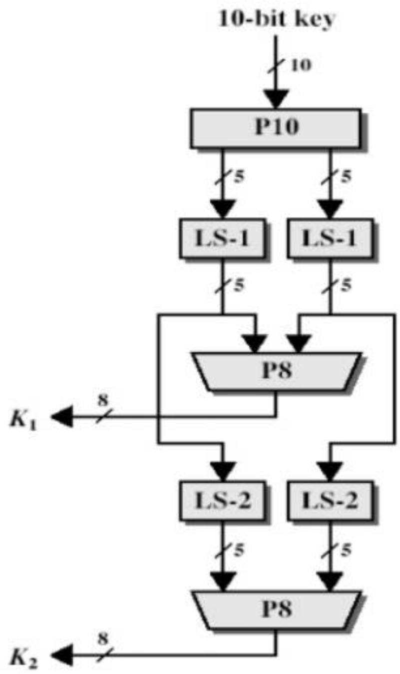
1. It is a block cipher.
2. It has 8-bits block size of plain text or cipher text.
3. It uses 10-bits key size for encryption.
4. It is a symmetric cipher.
5. It has Two Rounds.

The encryption algorithm involves five functions:

- an initial permutation (IP)
- a complex function labeled f_k, which involves both permutation and substitution operations and depends on a key input
- a simple permutation function that switches (SW) the two halves of the data
- the function f_k again
- a permutation function that is the inverse of the initial permutation

Key Generation Concept:

The function f_k takes as input not only the data passing through the encryption algorithm, but also an 8-bit key. Here a 10-bit key is used from which two 8-bit subkeys are generated. The key is first subjected to a permutation (P10). Then a shift operation is performed. The output of the shift operation then passes through a permutation function that produces an 8-bit output (P8) for the first subkey (K1). The output of the shift operation also feeds into another shift and another instance of P8 to produce the second subkey (K2).



Let the 10-bit key be designated as (k1, K2, k3, k4, k5, k6, k7, k8, k9, k10). Then the permutation P10 is defined as:

P10									
3	5	2	7	4	10	1	9	8	6

So the first output bit is bit 3 of the input; the second output bit is bit 5 of the input, and so on. For example, the key (1010000010) is permuted to (10000 01100). Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits.
Next we apply P8, which picks out and permutes 8 of the 10 bits according to the following rule:

P8							
6	3	7	4	8	5	10	9

The result is subkey 1 (K1). We then go back to the pair of 5-bit strings produced by the two LS-1 functions and performs a circular left shift of 2 bit positions on each string. In our example, the value (00001 11000) becomes (00100 00011). Finally, P8 is applied again to produce K2.

S-DES encryption

Encryption involves the sequential application of five functions.

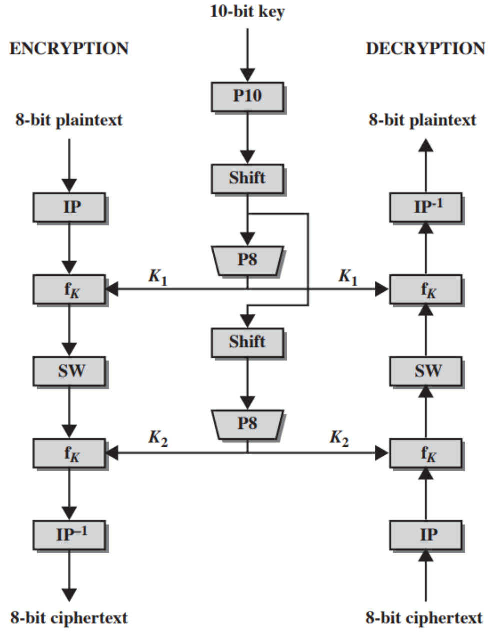


Figure G.1 Simplified DES Scheme

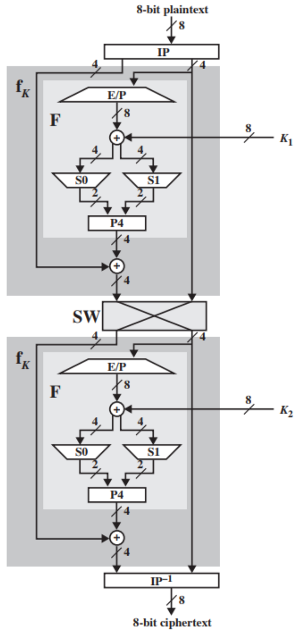


Figure G.3 Simplified DES Encryption Detail

- **Initial and Final Permutations**

The input to the algorithm is an 8-bit block of plaintext, which we first permute using the IP function:

IP							
2	6	3	1	4	8	5	7

The most complex component of S-DES is the function fK, which consists of a combination of permutation and substitution functions. The functions can be expressed as follows. Let L and R be the leftmost 4 bits and rightmost 4 bits of the 8-bit input to fK, and let F be a mapping (not necessarily one to one) from 4-bit strings to 4-bit strings. Then we let

$$f_K(L, R) = (L \oplus \mathrm{F}(R, SK), R)$$

Components in f_k are –

- **Expanded Permutation (EP)** –
  It takes a 4-bit input and converts it into an 8-bit output.

E/P							
4	1	2	3	2	3	4	1

The first 4 bits (first row of the preceding matrix) are fed into the S-box S0 to produce a 2- bit output, and the remaining 4 bits (second row) are fed into S1 to produce another 2-bit output.

These two boxes are defined as follows:

$$S0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \end{array} \qquad S1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} \end{array}$$

Next, the 4 bits produced by S0 and S1 undergo a further permutation as follows:

P4			
2	4	3	1

The output of P4 is the output of the function F.

- **The Switch Function**

The switch function (SW) interchanges the left and right 4 bits so that the second instance of f K operates on a different 4 bits. In this second instance, the E/P, S0, S1, and P4 functions are the same. The key input is K2. Finally apply inverse permutation to get the ciphertext.

Program

```
plaintext=input("Enter Plain Text:")
originalkey=input("Enter key")
pt=list(map(int,plaintext))
key=list(map(int,originalkey))
print("Plain Text:",pt)
print("10 bit Key:",key)
p10=[3,5,2,7,4,10,1,9,8,6]
p8=[6,3,7,4,8,5,10,9]
s0=[[1, 0, 3, 2],[3, 2, 1, 0],[0, 2, 1, 3],[3, 1, 3, 2]]
s1=[[0, 1, 2, 3 ],[2, 0, 1, 3],[3, 0, 1, 0 ],[2, 1, 0, 3 ]]
ip=[2,6,3,1,4,8,5,7]
ep=[4,1,2,3,2,3,4,1]
ip_inv=[4,1,3,5,7,2,8,6]
p4=[2,4,3,1]
kp10=[0]*len(p10)
m=len(key)//2
l1=[0]*m
l2=[0]*m
key1=[0]*len(p8)
key2=[0]*len(p8)
final=[0]*len(ip_inv)
de_final=[0]*len(ip_inv)
i_p=[0]*len(ip)
e_p=[0]*len(ep)
```

```
after_p4=[0]*len(p4)

def keygeneration():
    global l1,l2,key1,key2
    for i in range(0,len(key)):
        kp10[i]=key[p10[i]-1]
    l1=kp10[:m]
    l2=kp10[m:]
    last1,last2=l1[0],l2[0]
    for i in range(0,len(l1)-1):
        l1[i]=l1[i+1]
        l2[i]=l2[i+1]
    l1[-1],l2[-1]=last1,last2
    before_p81=l1.copy()
    before_p81.extend(l2)
    last1,sec_last1,last2,sec_last2=l1[0],l1[1],l2[0],l2[1]
    for i in range(0,len(l1)-2):
        l1[i]=l1[i+2]
        l2[i]=l2[i+2]
    l1[-1],l1[-2],l2[-1],l2[-2]=sec_last1,last1,sec_last2,last2
    before_p82=l1.copy()
    before_p82.extend(l2)
    for i in range(0,len(key1)):
        key1[i]=before_p81[p8[i]-1]
        key2[i]=before_p82[p8[i]-1]
    print("Key 1:",key1,"\nKey 2:",key2)

n=len(key1)//2
def generate_ip(text):
    for i in range(0,len(text)):
        i_p[i]=text[ip[i]-1]
    global l1,l2
    l1 = i_p[:n]
    l2=i_p[n:]

def pro(l1,l2,key):
    for i in range(0,len(pt)):
        e_p[i]=l2[ep[i]-1]
    if(key==1):
        skey=key1
    else:
        skey=key2
    ex=[a^b for a,b in zip(e_p,skey)]
    fors0=ex[:n]
    fors1=ex[n:]
    row_s0=str(fors0[0])+""+str(fors0[-1])
    col_s0=str(fors0[1])+""+str(fors0[2])
    row_s1=str(fors1[0])+""+str(fors1[-1])
```

```
    col_s1=str(fors1[1])+""+str(fors1[2])
    pos0=s0[int(row_s0,2)][int(col_s0,2)]
    pos1=s1[int(row_s1,2)][int(col_s1,2)]
    after_s0=list(bin(pos0).replace("0b",""))
    after_s1=list(bin(pos1).replace("0b",""))
    if(pos0==1 or pos0==0):
        after_s0.insert(0,'0')
    if(pos1==1 or pos1==0):
        after_s1.insert(0,'0')
    after_s0.extend(after_s1)
    after_p4=[0]*4
    for i in range(0,len(p4)):
        after_p4[i]=after_s0[p4[i]-1]
    after_p4=list(map(int,after_p4))
    exo=[a^b for a,b in zip(after_p4,l1)]
    exo.extend(l2)
    return exo

def encryption():
    global l1,l2,final
    generate_ip(pt)
    before_ep=pro(l1,l2,1)
    l1=before_ep[n:]
    l2=before_ep[:n]
    second_ep=pro(l1,l2,2)
    for i in range(0,len(ip_inv)):
        final[i]=second_ep[ip_inv[i]-1]
    print(final)
    ans="".join(map(str,final))
    answer=hex(int(ans,2)).replace("0x","")
    print("Cipher Text:",answer)

def decryption(final):
    global l1,l2
    generate_ip(final)
    before_ep=pro(l1,l2,2)
    l1=before_ep[n:]
    l2=before_ep[:n]
    second_ep=pro(l1,l2,1)
    for i in range(0,len(ip_inv)):
        de_final[i]=second_ep[ip_inv[i]-1]
    print(de_final)
    de_ans="".join(map(str,de_final))
    de_answer=hex(int(de_ans,2)).replace("0x","")
    print("Decrypted Text:",de_answer)

keygeneration()
encryption()
```

decryption(final)

Output

```
Plain Text: [1, 0, 0, 1, 0, 1, 1, 1]
10 bit Key: [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]
Key 1: [1, 0, 1, 0, 0, 1, 0, 0]
Key 2: [0, 1, 0, 0, 0, 0, 1, 1]
[0, 0, 1, 1, 1, 0, 0, 0]
Cipher Text: 38
[1, 0, 0, 1, 0, 1, 1, 1]
Decrypted Text: 97
```

Conclusion:

We understood the working of Simplified Data Encryption Standard (S-DES) algorithm how does it overcome the shortcomings of DES algorithm and implemented the steps stated by the S-DES algorithm to encrypt the data. While performing the experiment we studied each and every aspect of S-DES algorithm and the significance all the steps carried out during encryption and decryption of the data.

VAMOS Encryption and Decryption Algorithm

Vaishnavi Sawant
B.Tech. Student
Dept. of Computer Engineering
Dwarkadas J. Sanghvi College
of Engineering, Mumbai,
Maharashtra, India
vvaishsawant@gmail.com

Ahmed Solkar
B.Tech. Student
Dept. of Computer Engineering
Dwarkadas J. Sanghvi College
of Engineering, Mumbai,
Maharashtra, India
ahmedsolkar@gmail.com

Dr. Ramchandra Mangrulkar
Professor
Dept. of Computer Engineering
Dwarkadas J. Sanghvi College
of Engineering, Mumbai,
Maharashtra, India
ramchandra.mangrulkar@djsce.ac.in

***Abstract*—Internet (commonly called as Net) is a global network of billions of interconnected systems and other electronic devices. Internet makes it possible to transfer any form of data and facilitates data communication. This is done simply by connecting computer to the Internet. As Internet is public the data being transferred is vulnerable to a lot of data theft attacks such as packet sniffing, packet spoofing etc., it is essential to protect the private data from such kind of attacks. When the network is not secure hackers can exploit the connection, giving them access to sensitive information and can possibly tamper the data.**

In today's world data is very crucial for businesses and individuals, hence it must be protected from all kinds of violations. Data must be secured from unauthorized access and manipulation. It is challenging to physically secure the network so data encryption can provide significant amount of security. Encryption is a technique through which the data is transformed in some incomprehensible form which cannot be interpreted by any attacker. After successful transmission of data, the receiver can restore the data received into its original form by applying appropriate decryption technique.

In this paper, we have presented a symmetric data encryption algorithm developed in Python to encrypt images, making the transmission secure.

***Index Terms*—Cryptography, Encryption, Decryption, Internet, Python**

I. Introduction

The technique through which information is transmitted in a secured way and communication is conducted via some programs and procedures so that only the intended entity understands the actual data that is shared is known as Cryptography. This process thwarts unauthorized accessibility for the information. During data transmission across the network, Cryptography aims to fulfil four different objectives:

- **Confidentiality:** This ensures that only the intended receiver is able to decrypt the received message and access the original data.
- **Non-repudiation:** Non-repudiation ensures that the sender of the message cannot deny their participation in the communication by sending or creating the message.
- **Integrity:** Integrity focuses on ensuring that the information is not modified or compromised in any way while in storage or transit.
- **Authenticity:** Authenticity ensures that the sender and receiver can verify each other's identities and check whether the information came from legitimate source.

Encryption is a technique that is used to conceal the data that is to be transmitted. This is done by applying different algorithms which consist of numerous mathematical and logical computations. The resultant data is called as 'Cipher Text'. The original data commonly known as 'Plain Text' along with some 'Secret Key' is supplied to an algorithm which does the job of encrypting the data. At the receiver's side this 'Cipher Text' is reverted back to its original form by applying the appropriate decryption algorithm and the 'Secret Key'. Encryption has two types:

Symmetric Encryption: When only one 'Secret Key' is used for encrypting and decrypting the data it is known as Symmetric Encryption. The same Key is distributed among all the communicating parties in the network. For e.g. DES Algorithm, AES algorithm etc.

Asymmetric Encryption: In this two keys are used for the process of encryption and decryption. For Encryption the algorithm uses the 'Public Key' of the intended receiver which is shared publicly, while for Decryption it uses the 'Private Key' of the receiver which is kept private. For e.g RSA Algorithm, Elliptical Curve Cryptography etc.

II. Methodology used

A. Overview

The presented algorithm is based on symmetric encryption technique and incorporates the following computations: Circular Left Shift: Shifts the bits of the first operand by number of bits specified by the right operand such that the bits which fall off at one end are appended to the other.

Ex-OR: It is a logical operation also known as exclusive or which takes two Boolean operands and returns true if, and only if, the operands are different.

Mix Columns: It is matrix multiplication similar to AES . It ha s a predefined 'Multiplication Matrix' which is multiplied with Input 4x4 matrix. The results of these multiplications are XORed together to produce only 4 result bytes for the next state. Therefore it contains 4 bytes input, 16 multiplications 12 XORs and 4 bytes output. This is a complex computation

and hence it is simplified by performing it over a Galois Field. Two tables named ‘E’ and ‘L’ are made for this and the result of the multiplication is simply the result of a lookup of the L table, followed by the addition of the results, followed by a lookup to the E table.
Shift Rows: The entire row of the input matrix is relocated to some other index in the same matrix based on some criteria.
S-BOX: Substitution Box is a 16x16 invertible matrix similar to AES encryption technique. It transforms the 8 but input data into 8 bit secret data using a precomputed Look Up table. This table provides confusion in the Cipher Text by substituting the some in place of original values. the design of S-BOX is used to protect the message and also achieve a high throughput , high energy efficiency and occupy less area.

B. Flowchart

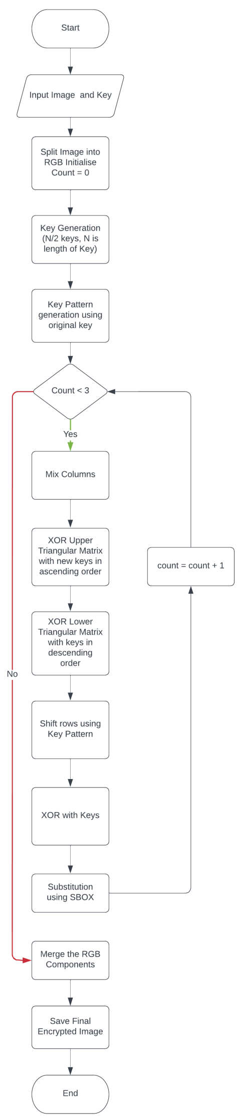


Fig. 1: Encryption Flowchart.

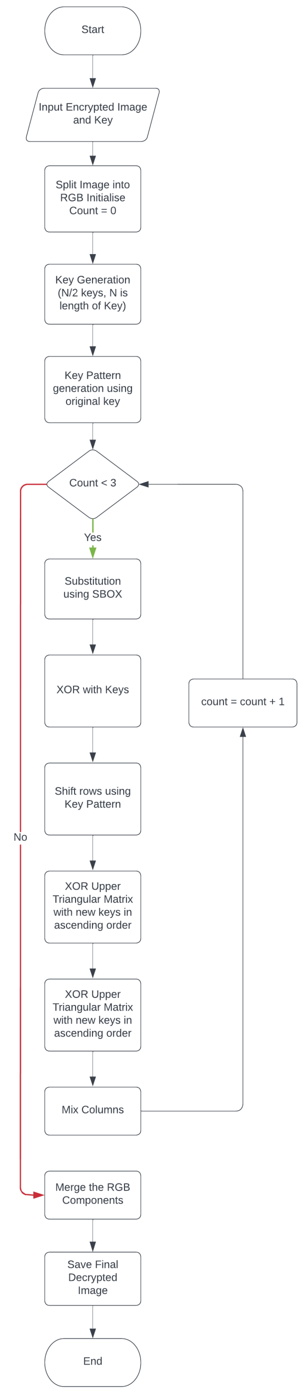


Fig. 2: Decryption Flowchart.

C. Algorithm

Step 1: Take Image to be transmitted and Key used for Encryption from the User.
Step 2: Split the Input Image into RGB matrices.
Step 3: Key Generation
  3.1: Left Shift by 2
  3.2: Swap the first half with the second half
  3.3: Ex-OR with opposite index elements.
  3.4: Apply P8 combination to generate final n/2 keys.
Step 4: Generate Key Pattern.
Step 5: Perform Mix Columns for every 4x4 matrix.
Step 6: Ex-OR Upper Triangular Matrix of the obtained result with the keys in ascending order.
Step 7: Ex-OR Lower Triangular Matrix of the obtained result with the keys in descending order.
Step 8: Shift rows of the resultant matrix according to the pattern generated in Step 4.
Step 9: Ex-OR each element with generated keys.
Step 10: Perform S-BOX substitution.
Step 11: Merge the RGB matrices and save the encrypted image.

III. Working

The presented algorithm is based on Symmetric Key Cryptography which means that it uses only one key for encryption as well as for decryption, which is taken as input from the user along with the image to be encrypted. The algorithm begins with splitting the image in 3 matrices each containing the Red, Green and Blue values of each pixel of the image respectively. All the computations included in the algorithm are applied to each of these matrices separately and the resultant matrices are merged to form the final encrypted image. Same procedure is followed while Decryption.

After the image is split the next step is Key Generation. In this step, n/2 keys are derived from the original key by performing a series of computations. The characters of the input text Key are firstly converted into their equivalent binary value and Left Shift by 2 operation is applied to all these values individually followed by swapping of their equal length sections. The resultant values are then XOR ed with their negative index values thus deriving the n/2 keys from length n Key. Each of these n/2 keys are permuted by application of p8 table and the final form of n/2 keys is achieved.

Along with this a Key Pattern based on the ASCII values of the characters of the input Key is extracted. This pattern is applied further while Shift Rows transformation.

Further to provide diffusion in the data image matrix we apply MixColumns transformation similar to AES. This is done for interbyte transformation that changes the bits inside a byte. This transformation changes the content of every byte by taking four bytes at a time and combining them to create new four bytes. It uses a polynomial function which takes four bytes of one input column and outputs four new bytes which replace the original ones.

For applying MixColumns to our input matrix we first split the original matrix into number of 4x4 matrices as MixColumns transformation works on 4x4 matrix. These 4x4 matrices are given as input to the MixColumns one by one. MixColumns transformation uses a predefined ‘Multiplication Matrix’ for its computation. This multiplication is performed one column at a time for each 4x4 input matrix. Eventually each value in the input column is multiplied with every value of the ‘Multiplication Matrix’ resulting in total 16 multiplications for a single column. The resultant values of the multiplication are XOR ed together to produce 4 bytes which replace the original column. Hence for each column there are 4 bytes input, 16 multiplications 12 XOR’s giving a 4 bytes output. This multiplication is performed one row at a time for each value of the input column.

The Multiplication Matrix for Encryption is given as:

TABLE I: Multiplication Matrix

2	3	1	1
1	2	3	1
1	1	2	3
3	1	1	2

If the 4x4 input matrix is given as:

TABLE II: Input Matrix

B1	B5	B9	B13
B2	B6	B10	B14
B3	B7	B11	B15
B4	B8	B12	B16

Then the resultant value of B1 is calculated by multiplying the 4 values of the input column with the first row values of the ‘Multiplication Matrix’ and then every value is Ex-OR ed with the other.

B1=(B1*2) xor (B2*3) xor (B3*1) xor (B4*1)

Similarly the resultant value of B2 is calculated by multiplying the 4 values of the input column with the second row values of the ‘Multiplication Matrix’ and then every value is Ex-OR ed with the other.

B2=(B1*1) xor (B2*2) xor (B3*3) xor (B4*1)

However this computation is complex and takes up a lot of processing time hence an alternative known as ‘Galois Field Multiplication’ is implemented in the presented algorithm. In this technique lookup tables named ‘L table’ and ‘E table’ are defined for ease of multiplication. The result of multiplication is derived by referring the L table followed by the addition (regular mathematical addition) of these results and then referring the ‘E table’. [1]

For example:
Consider Input = D4 BF 5D 30
Output (0) = (D4 *2) xor (BF * 3) xor (5D * 1) xor (30 *1)
= E (L (D4) + L (02)) xor E(L (BF) + L (03)) xor 5D xor 30
= E (41 + 19) xor E (9D + 01) xor 5D xor 30
= E (5A) xor E (9E) xor 5D xor 30
= B3 xor DA xor 5D xor 30
= 04

Following the application of MixColumns transformation on all the 4x4 matrices generated from the original matrix , all these resultant matrices are merged into a single matrix

After the application of MixColumns transformation we perform row- wise Ex-Or operation on the partially encrypted matrix which is divided into two parts (i.e. upper-triangular and lower-triangular). In upper triangular matrix the keys are Ex-OR ed with the values present in the row in an ascending order. While in lower triangular matrix the keys are Ex-OR ed with the values present in the row in a descending order.

The prior step is then followed by Shift Rows operation. In this the rows of resultant matrix are shuffled according to the Key Pattern generated previously.

After the shuffling, all elements of the matrix are Ex-OR ed with the keys generated in a sequential way such that each element is Ex-OR ed with one particular key.

Finally the elements of the matrix undergo substitution using a precomputed Look-Up Table named S-BOX (Substitution Bytes). S-BOX is a 16x16 matrix through which each element (byte) of the matrix is substituted with some other byte by referring the pre-computed S-BOX table.

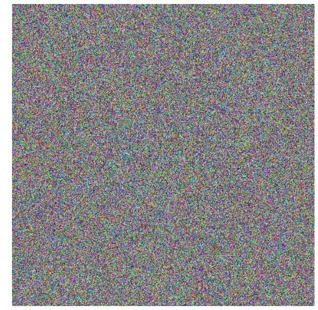
IV. Histogram Analysis

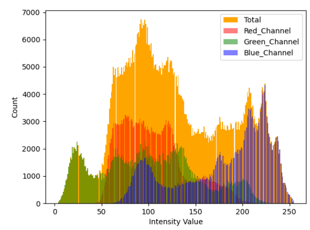
Histogram of an image is used to show the distribution of intensity values of the pixels of that image. For most Images, when we plot the histogram the intensity values are not uniformly distributed. Although, when the same Image is encrypted using certain Encryption algorithm it should have a histogram which is uniformly distributed or near equal intensity distribution in order to defend against various statistical attacks.

Now, we use standard Lena Image as our Source Image Fig 3(a). When performing Histogram analysis of the source Image, we can see it has a highly variable intensity distribution. Upon encrypting using the proposed Encryption Algorithm, the encrypted image and its following Histogram Analysis is shown in Fig 3(b) and Fig 3(c) respectively.

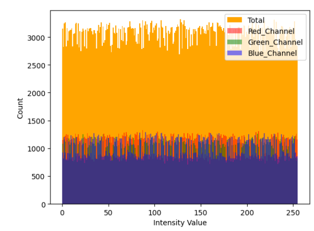
It can be observed that the encrypted Image is almost uniformly Distributed. The Original Lena Image has highly varied intensity distribution ranging from 1000 to 6,800. The intensity distribution of Encrypted Image ranges from 2800-3100. This reveals that the intensity distribution of the encrypted image is much more uniform than the Original Image. Thus, the proposed algorithm brings a good amount of uniformity in terms of intensity distribution. This observation implies that the proposed algorithm will provide a good level of defense and security against different statistical attacks.


(a) Source Image of Lena


(b) Encrypted Image



(c) Histogram of Original Image



(d) Histogram of Encrypted Image

Fig. 3: Histogram Analysis

V. Conclusions

In this paper we have presented an image encryption algorithm which is based on Symmetric Key Cryptography. In order to enhance the security and confidentiality of the information new keys are derived from the Secret Key shared between the communicating parties and these new keys are used in computations instead of the original key hence reducing the influence of Man-In-Middle attack. To enhance the final encrypted image making it difficult to decipher, the presented algorithm splits the image pixels into 3 matrices each containing Red, Green and Blue values of the pixels respectively and applies all the computations to each of these matrices separately. These computations consist of low-level operations such as bit level Ex-OR, addition, multiplication, various permutation, substitution and shifting operations. The algorithm consists 2 modules first is of Key Generation which derives novel keys to be used in the algorithm, it also generates a pattern from the Key based on some criteria which is used later in the application of algorithm. Second module consists of all the computations to be applied on the input matrices. These computations are composed of following major steps 1) Mix-Columns Transformation, 2) Upper and Lower Triangular Ex-Or, 3)Shift Rows Transformation, 4) Element-wise Ex-Or and 5) Sub-Bytes Transformation. The resultant matrices are merged into one in order to produce the final encrypted image. This image along with the Shared Secret Key is transmitted to the recipient who decrypts the image. The decryption process is the total reverse of the encryption process. At the recipient The Key Generation module is executed first generating the keys similar to the encryption process and all the other computations are applied in a reverse order to get the Original image. The presented algorithm provides a strong encrypted format of the image which can also be decrypted without any data loss and errors. Hence it provides a secure and efficient means for transfer of images in real time.

REFERENCES

[1] https://www.infosecwriters.com/textresources/pdf/AESbyExample.pdf

Experiment 5

Date of Performance : 21-04-2022 **Date of Submission: 03-06-2022**

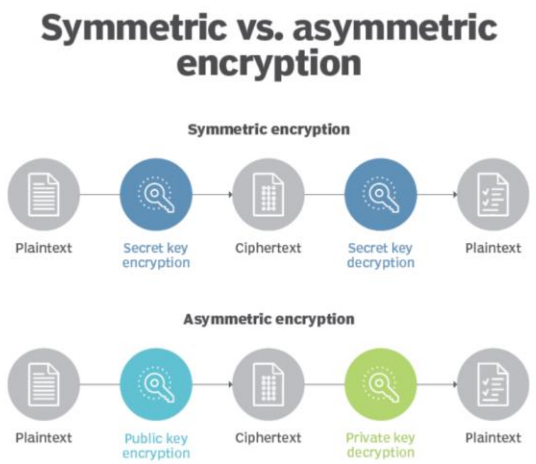
SAP Id: 60004200059 **Name : Vaishnavi Sawant**
Div: A **Batch : A2**

Aim of Experiment

Implement RSA Algorithm

Theory:

The RSA algorithm is an asymmetric cryptography algorithm; this means that it uses a public key and a private key (i.e two different, mathematically linked keys). As their names suggest, a public key is shared publicly, while a private key is secret and must not be shared with anyone.The RSA algorithm is named after those who invented it in 1978: Ron Rivest, Adi Shamir, and Leonard Adleman.
In RSA cryptography, both the public and the private keys can encrypt a message. The opposite key from the one used to encrypt a message is used to decrypt it. This attribute is one reason why RSA has become the most widely used asymmetric algorithm: It provides a method to assure the confidentiality, integrity, authenticity, and non-repudiation of electronic communications and data storage.



The option to encrypt with either the private or public key provides a multitude of services to RSA users. If the public key is used for encryption, the private key must be used to decrypt the data. This is perfect for sending sensitive information across a network or Internet connection, where the recipient of the data sends the data sender their public key. The sender of the data then encrypts the sensitive information with the public key and sends it to the recipient. Since the public key encrypted the data, only the owner of the private key can decrypt the sensitive data. Thus, only the intended recipient of the data can decrypt it, even if the data were taken in transit.
The other method of asymmetric encryption with RSA is encrypting a message with a private key. In this example, the sender of the data encrypts the data with their private key and sends encrypted data and their public key along to the recipient of the data. The recipient of the data

can then decrypt the data with the sender's public key, thus verifying the sender is who they say they are. With this method, the data could be stolen and read in transit, but the true purpose of this type of encryption is to prove the identity of the sender. If the data were stolen and modified in transit, the public key would not be able to decrypt the new message, and so the recipient would know the data had been modified in transit.

Many protocols, including Secure Shell (SSH), OpenPGP, S/MIME, and SSL/TLS, rely on RSA for encryption and digital signature functions. It is also used in software programs -- browsers are an obvious example, as they need to establish a secure connection over an insecure network, like the internet, or validate a digital signature. RSA signature verification is one of the most commonly performed operations in network-connected systems.

Steps in RSA:

- Select two large prime numbers, p and q.
- Multiply these numbers to find n = p x q, where n is called the modulus for encryption and decryption.
- Choose a number e less than n, such that n is relatively prime to (p - 1) x (q -1). It means that e and (p - 1) x (q - 1) have no common factor except 1. Choose "e" such that 1<e < φ (n), e is prime to φ (n),
  gcd (e,d(n)) =1
- If n = p x q, then the public key is <e, n>. A plaintext message m is encrypted using public key <e, n>. To find ciphertext from the plain text following formula is used to get ciphertext C.
  $C = m^e \bmod n$
  Here, m must be less than n. A larger message (>n) is treated as a concatenation of messages, each of which is encrypted separately.
- To determine the private key, we use the following formula to calculate the d such that:
  $D_e \bmod \{(p - 1) \times (q - 1)\} = 1$
  Or
  $D_e \bmod \varphi (n) = 1$
- The private key is <d, n>. A ciphertext message c is decrypted using private key <d, n>. To calculate plain text m from the ciphertext c following formula is used to get plain text m.
  $m = c^d \bmod n$

Example:

where p = 17 and q=13. Value of e can be 5 as it satisfies the condition 1 < e < (p-1)(q-1).
N = p * q = 91
D = e-1mod(p-1)(q-1) = 29
Public Key pair = (91,5)
Private Key pair = (91,29)
If the plaintext(m) value is 10, you can encrypt it using the formula me mod n = 82.
To decrypt this ciphertext(c) back to original data, you must use the formula cd mod n = 29.

Advantages of RSA

- No Key Sharing: RSA encryption depends on using the receiver's public key, so you don't have to share any secret key to receive messages from others.

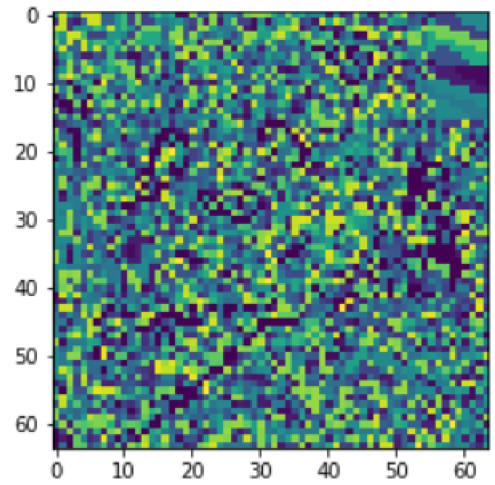
- Proof of Authenticity: Since the key pairs are related to each other, a receiver can't intercept the message since they won't have the correct private key to decrypt the information.
- Faster Encryption: The encryption process is faster than that of the DSA algorithm.
- Data Can't Be Modified: Data will be tamper-proof in transit since meddling with the data will alter the usage of the keys. And the private key won't be able to decrypt the information, hence alerting the receiver of manipulation.

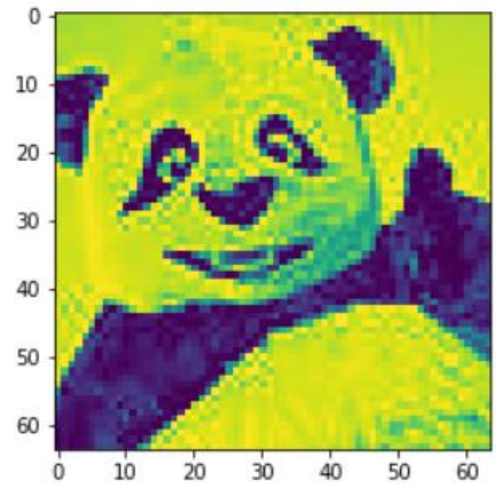
Code:

```
import math
import numpy as np
import PIL
import matplotlib.pyplot as plt

p,q=229,41
n=p*q
f_n=(p-1)*(q-1)
e=2

def mul_Inverse(a,b):
    ri = [b,a]
    qi=[0,0]
    xi=[1,0]
    yi=[0,1]
    i=2
    while True:
        ri.append(ri[i-2]%ri[i-1])
        qi.append(ri[i-2]//ri[i-1])
        xi.append(xi[i-2]-qi[i]*xi[i-1])
        yi.append(yi[i-2]-qi[i]*yi[i-1])
        if ri[i]==1:
            break
        i+=1

    XI=(b+xi[-1])if(xi[-1]<0)else(xi[-1])
    YI=(b+yi[-1])if(yi[-1]<0)else(yi[-1])
    return YI

def encryp(m):
    global e,n,f_n
    f=0
    encr=0
    while(f==0):
        g=math.gcd(e,f_n)
        if(g==1):
            f=1
        else:
            e+=1
```

```
    encr=(m**e)%n
    return encr

def decryp(en):
    global e,f_n,n
    d=mul_Inverse(e,f_n)
    pt=(en**d)%n
    return pt

def image_encrypt(img):
    gray_img = img.convert("L")
    imgArr = np.array(gray_img)
    flatten_array=imgArr.flatten()
    shape=imgArr.shape
    new_arr=list()
    for i in range(0,len(flatten_array)):
        enc=encryp(int(flatten_array[i]))
        new_arr.append(enc)
    show_arr=np.reshape(new_arr,(64,64))
    plt.imshow(show_arr)


def image_decrypt(dimg):
    flat_dec_array=dimg.flatten()
    dec_arr=list()
    for i in range(0,len(flat_dec_array)):
        dec=decryp(int(flat_dec_array[i]))
        dec_arr.append(dec)
    showd_arr=np.reshape(dec_arr,(64,64))
    plt.imshow(showd_arr)



img = PIL.Image.open('images_sm.jpg')
image_encrypt(img)
image_decrypt(show_arr)
```

Output:





Conclusion:

Hence through this experiment we understood the importance of asymmetric cryptography, the workflow in RSA, the steps involved in the signature verification, and the perks it offers over other standards

Experiment 6

Date of Performance : 28-04-2022 **Date of Submission: 03-06-2022**

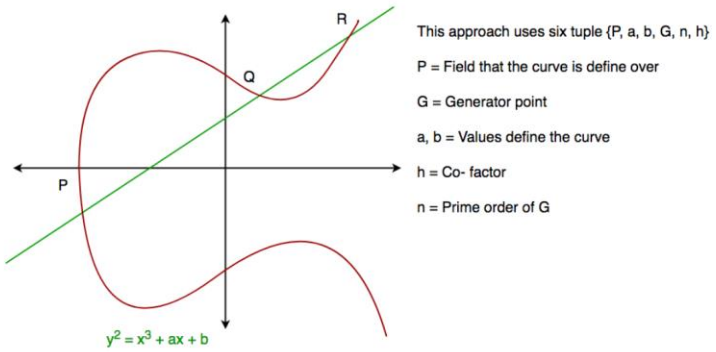
SAP Id: 60004200059 **Name : Vaishnavi Sawant**
Div: A **Batch : A2**

Aim of Experiment

Implement Diffie Hellman Algorithm

Theory:

The Diffie–Hellman (DH) Algorithm is a key-exchange protocol that enables two parties communicating over public channel to establish a mutual secret without it being transmitted over the Internet. DH enables the two to use a public key to encrypt and decrypt their conversation or data using symmetric cryptography.
The algorithm is based on Elliptic Curve Cryptography, a method of doing public-key cryptography based on the algebra structure of elliptic curves over finite fields. The DH also uses the trapdoor function, just like many other ways to do public-key cryptography.



The simple idea of understanding to the DH Algorithm is the following.

1. The first party picks two prime numbers, g and p and tells them to the second party.
2. The second party then picks a secret number (let's call it a), and then it computes g^a mod p and sends the result back to the first party; let's call the result A. Keep in mind that the secret number is not sent to anyone, only the result is.
3. Then the first party does the same; it selects a secret number b and calculates the result B i.e. g^b mod p.
4. Then, this result is sent to the second party.
5. The second party takes the received number B and calculates B^a mod p
6. The first party takes the received number A and calculates A^b mod p

The number we came within steps 5 and 6 will be taken as the shared secret key. This key can be used to do any encryption of data that will be transmitted, such as blowfish, AES, etc.
P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

Example

1. Alice and Bob both use public numbers P = 23, G = 5
2. Alice selected private key a = 4, and Bob selected b = 3 as the private key
3. Both Alice and bob now calculate the value of x and y as follows:

- Alice: $x = (5^4 \bmod 23) = 4$
- Bob: $y = (5^3 \bmod 23) = 10$

4. Now, both Alice and Bob exchange public numbers with each other.
5. Alice and Bob now calculate the symmetric keys

- Alice: $k_a = y^a \bmod p = 10^4 \bmod 23 = 18$
- Bob: $k_b = x^b \bmod p = 4^3 \bmod 23 = 18$

6. 18 is the shared secret key.

<u>Advantages of the Diffie Hellman Algorithm</u>

- The sender and receiver don't need any prior knowledge of each other.
- Once the keys are exchanged, the communication of data can be done through an insecure channel.
- The sharing of the secret key is safe.

<u>Disadvantages of the Diffie Hellman Algorithm</u>

- The algorithm can not be sued for any asymmetric key exchange.
- Similarly, it can not be used for signing digital signatures.
- Since it doesn't authenticate any party in the transmission, the Diffie Hellman key exchange is susceptible to a man-in-the-middle attack.

Code:

```
import math
def check_prime(p):
    for i in range(2,p//2):
        r=p%i
        if(r==0):
            return 0
    return 1
def check_primitiveroot(p,g):
    gc=math.gcd(g,p)
    if(gc==1):
        return 1
    else:
        return 0

def generate_secret():
    p=int(input("Enter p (Prime number)"))
    g=int(input("Enter g (Primitive Root of p)"))
    if(check_prime(p)==1 and check_primitiveroot(p,g)==1):
        a=4
```

```
    b=3

    Alice_x=(g**a)%p
    Bob_y=(g**b)%p

    Alice_secret=(Bob_y**a)%p
    Bob_secret=(Alice_x**b)%p

    print("Secret Key:",Alice_secret)
def Encrypt(plainText,key):
  ciphertext = ""
  for i in range(0,len(plainText)):
    ch = plainText[i]
    if ch==' ':
      ciphertext+=" "
    elif ch.isnumeric():
      asci_val = (ord(ch) - 48 +key)%10
      ciphertext+= chr(48 + asci_val)
    else:
      asci_val = (ord(ch)-65+key)%26
      ciphertext+=chr(65+asci_val)

  return ciphertext

def Decrypt(cipherText, key):
  plainText = ""
  for i in range(0, len(cipherText)):
    ch = cipherText[i]
    if ch == " ":
      plainText += " "
    elif ch.isnumeric():
      if key > 9:
        key %= 10
      ascii_val = (ord(ch) - 48 - key)
      if ascii_val < 0:
        ascii_val =  abs(ascii_val) % 10
        plainText += chr(57 - ascii_val + 1)
      else:
        ascii_val =  abs(ascii_val) % 10
        plainText += chr(48 + ascii_val)
    else:
      if key > 25:
        key %= 26
      ascii_val = (ord(ch) - 65 - key)
      if ascii_val < 0:
        ascii_val =  abs(ascii_val) % 26
        plainText += chr(90 - ascii_val + 1)
      else:
```

```
            ascii_val =  abs(ascii_val) % 26
            plainText += chr(65 + ascii_val)
    print("Decrypted Text at Bob's end: ", plainText)

pt=input("Enter plain text for encryption:")
generate_secret()
cipher = Encrypt(pt.upper(),Alice_secret)
print("Cipher Text from Alice:",cipher)
Decrypt(cipher,Bob_secret)
```

Output:

```
Secret Key: 9
Cipher Text from Alice: MROORN QNUUVJW
Decrypted Text at Bob's end:  DIFFIE HELLMAN
```

Conclusion:

Thus we understood how Diffie–Hellman (DH) Algorithm enables two parties communicating over public channel to establish a mutual secret without it being transmitted over the Internet and secures the transmission of data.

Experiment 7

Date of Performance : **05-05-2022** **Date of Submission:** **03-06-2022**

SAP Id: 60004200059 **Name : Vaishnavi Sawant**
Div: A **Batch : A2**

Aim of Experiment

Study the use of network reconnaissance tools like WHOIS, dig, traceroute, nslookup to gather information about networks and domain registrars.

Theory:

whois:

whois command is used to find out the information about a domain, such as the owner of the domain, the owner's contact information, and the name servers that the domain is using.It allows you to perform lookup of owner information of a website by querying databases that store the registered users of a domain or IP address.
whois is a command that searches the "who is" database for information on the owner of a particular domain name. WHOIS is a query and response protocol that is widely used for querying databases that store the registered users of an Internet resource, such as a domain name or an IP address block, but is also used for a wider range of other information. The information provided can include the contact name, address, email address and phone number. The whois command will also return the name servers and certain status information.

Most modern versions of whois try to guess the right server to ask for the specified object. If no guess can be made, whois will connect to whois.networksolutions.com for NIC handles or whois.arin.net for IPv4 addresses and network names.

```
D:\whois>whois google.com

Whois v1.21 - Domain information lookup
Copyright (C) 2005-2019 Mark Russinovich
Sysinternals - www.sysinternals.com

Connecting to COM.whois-servers.net...

WHOIS Server: whois.markmonitor.com
   Registrar URL: http://www.markmonitor.com
   Updated Date: 2019-09-09T15:39:04Z
   Creation Date: 1997-09-15T04:00:00Z
   Registry Expiry Date: 2028-09-14T04:00:00Z
   Registrar: MarkMonitor Inc.
   Registrar IANA ID: 292
   Registrar Abuse Contact Email: abusecomplaints@markmonitor.com
   Registrar Abuse Contact Phone: +1.2086851750
   Domain Status: clientDeleteProhibited https://icann.org/epp#clientDeleteProhibited
   Domain Status: clientTransferProhibited https://icann.org/epp#clientTransferProhibited
   Domain Status: clientUpdateProhibited https://icann.org/epp#clientUpdateProhibited
   Domain Status: serverDeleteProhibited https://icann.org/epp#serverDeleteProhibited
   Domain Status: serverTransferProhibited https://icann.org/epp#serverTransferProhibited
   Domain Status: serverUpdateProhibited https://icann.org/epp#serverUpdateProhibited
   Name Server: NS1.GOOGLE.COM
   Name Server: NS2.GOOGLE.COM
   Name Server: NS3.GOOGLE.COM
   Name Server: NS4.GOOGLE.COM
   DNSSEC: unsigned
   URL of the ICANN Whois Inaccuracy Complaint Form: https://www.icann.org/wicf/
>>> Last update of whois database: 2022-05-12T13:38:33Z <<<

For more information on Whois status codes, please visit https://icann.org/epp

NOTICE: The expiration date displayed in this record is the date the
registrar's sponsorship of the domain name registration in the registry is
currently set to expire. This date does not necessarily reflect the expiration
date of the domain name registrant's agreement with the sponsoring
registrar.  Users may consult the sponsoring registrar's Whois database to
view the registrar's reported date of expiration for this registration.

TERMS OF USE: You are not authorized to access or query our Whois
database through the use of electronic processes that are high-volume and
automated except as reasonably necessary to register domain names or
modify existing registrations; the Data in VeriSign Global Registry
Services' ("VeriSign") Whois database is provided by VeriSign for
information purposes only, and to assist persons in obtaining information
about or related to a domain name registration record. VeriSign does not
guarantee its accuracy. By submitting a Whois query, you agree to abide
by the following terms of use: You agree that you may use this Data only
```

```
for lawful purposes and that under no circumstances will you use this Data
to: (1) allow, enable, or otherwise support the transmission of mass
unsolicited, commercial advertising or solicitations via e-mail, telephone,
or facsimile; or (2) enable high volume, automated, electronic processes
that apply to VeriSign (or its computer systems). The compilation,
repackaging, dissemination or other use of this Data is expressly
prohibited without the prior written consent of VeriSign. You agree not to
use electronic processes that are automated and high-volume to access or
query the Whois database except as reasonably necessary to register
domain names or modify existing registrations. VeriSign reserves the right
to restrict your access to the Whois database in its sole discretion to ensure
operational stability.  VeriSign may restrict or terminate your access to the
Whois database for failure to abide by these terms of use. VeriSign
reserves the right to modify these terms at any time.

The Registry database contains ONLY .COM, .NET, .EDU domains and
Registrars.

Connecting to whois.markmonitor.com...

WHOIS Server: whois.markmonitor.com
Registrar URL: http://www.markmonitor.com
Updated Date: 2019-09-09T15:39:04+0000
Creation Date: 1997-09-15T07:00:00+0000
Registrar Registration Expiration Date: 2028-09-13T07:00:00+0000
Registrar: MarkMonitor, Inc.
Registrar IANA ID: 292
Registrar Abuse Contact Email: abusecomplaints@markmonitor.com
Registrar Abuse Contact Phone: +1.2083895770
Domain Status: clientUpdateProhibited (https://www.icann.org/epp#clientUpdateProhibited)
Domain Status: clientTransferProhibited (https://www.icann.org/epp#clientTransferProhibited)
Domain Status: clientDeleteProhibited (https://www.icann.org/epp#clientDeleteProhibited)
Domain Status: serverUpdateProhibited (https://www.icann.org/epp#serverUpdateProhibited)
Domain Status: serverTransferProhibited (https://www.icann.org/epp#serverTransferProhibited)
Domain Status: serverDeleteProhibited (https://www.icann.org/epp#serverDeleteProhibited)
Registrant Organization: Google LLC
Registrant State/Province: CA
Registrant Country: US
Registrant Email: Select Request Email Form at https://domains.markmonitor.com/whois/google.com
Admin Organization: Google LLC
Admin State/Province: CA
Admin Country: US
Admin Email: Select Request Email Form at https://domains.markmonitor.com/whois/google.com
Tech Organization: Google LLC
Tech State/Province: CA
Tech Country: US
Tech Email: Select Request Email Form at https://domains.markmonitor.com/whois/google.com
Name Server: ns2.google.com
Name Server: ns1.google.com
```

```
Name Server: ns4.google.com
Name Server: ns3.google.com
DNSSEC: unsigned
URL of the ICANN WHOIS Data Problem Reporting System: http://wdprs.internic.net/
>>> Last update of WHOIS database: 2022-05-12T13:29:20+0000 <<<

For more information on WHOIS status codes, please visit:
  https://www.icann.org/resources/pages/epp-status-codes

If you wish to contact this domainÔÇÖs Registrant, Administrative, or Technical
contact, and such email address is not visible above, you may do so via our web
form, pursuant to ICANNÔÇÖs Temporary Specification. To verify that you are not a
robot, please enter your email address to receive a link to a page that
facilitates email communication with the relevant contact(s).

Web-based WHOIS:
  https://domains.markmonitor.com/whois

If you have a legitimate interest in viewing the non-public WHOIS details, send
your request and the reasons for your request to whoisrequest@markmonitor.com
and specify the domain name in the subject line. We will review that request and
may ask for supporting documentation and explanation.

The data in MarkMonitorÔÇÖs WHOIS database is provided for information purposes,
and to assist persons in obtaining information about or related to a domain
nameÔÇÖs registration record. While MarkMonitor believes the data to be accurate,
the data is provided "as is" with no guarantee or warranties regarding its
accuracy.

By submitting a WHOIS query, you agree that you will use this data only for
lawful purposes and that, under no circumstances will you use this data to:
  (1) allow, enable, or otherwise support the transmission by email, telephone,
or facsimile of mass, unsolicited, commercial advertising, or spam; or
  (2) enable high volume, automated, or electronic processes that send queries,
data, or email to MarkMonitor (or its systems) or the domain name contacts (or
its systems).

MarkMonitor reserves the right to modify these terms at any time.

By submitting this query, you agree to abide by this policy.

MarkMonitor Domain Management(TM)
Protecting companies and consumers in a digital world.

Visit MarkMonitor at https://www.markmonitor.com
Contact us at +1.8007459229
In Europe, at +44.02032062220
--
```

dig:

The **dig** (domain information groper) command is a flexible tool for interrogating DNS name servers. It performs DNS lookups and displays the answers that are returned from the queried name server(s). Dig tool is more flexible and better DNS tool than the Windows NSLookup tool. The dig command, allows you to query information about various DNS records, including host addresses, mail exchanges, and name servers. Most DNS administrators use the **dig** command to troubleshoot DNS problems because of its flexibility, ease of use, and clarity of output.

```
C:\WINDOWS\system32>dig google.com

; <<>> DiG 9.16.28 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 32122
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.             97      IN      A       172.217.160.206

;; Query time: 3 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Thu May 12 19:31:19 India Standard Time 2022
;; MSG SIZE  rcvd: 55


C:\WINDOWS\system32>
```

```
C:\WINDOWS\system32>dig 192.168.0.1

; <<>> DiG 9.16.28 <<>> 192.168.0.1
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 3188
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;192.168.0.1.                   IN      A

;; AUTHORITY SECTION:
.                       86398   IN      SOA     a.root-servers.net. nstld.verisign-grs.com. 2022051200 1800 900 604800 86400

;; Query time: 5 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Thu May 12 19:38:41 India Standard Time 2022
;; MSG SIZE  rcvd: 115
```

```
C:\WINDOWS\system32>dig

; <<>> DiG 9.16.28 <<>>
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21734
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;.                              IN      NS

;; ANSWER SECTION:
.                       84877   IN      NS      m.root-servers.net.
.                       84877   IN      NS      b.root-servers.net.
.                       84877   IN      NS      c.root-servers.net.
.                       84877   IN      NS      d.root-servers.net.
.                       84877   IN      NS      e.root-servers.net.
.                       84877   IN      NS      f.root-servers.net.
.                       84877   IN      NS      g.root-servers.net.
.                       84877   IN      NS      h.root-servers.net.
.                       84877   IN      NS      a.root-servers.net.
.                       84877   IN      NS      i.root-servers.net.
.                       84877   IN      NS      j.root-servers.net.
.                       84877   IN      NS      k.root-servers.net.
.                       84877   IN      NS      l.root-servers.net.

;; Query time: 4 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Thu May 12 19:39:42 India Standard Time 2022
;; MSG SIZE  rcvd: 239
```

nslookup:

nslookup (stands for "Name Server Lookup") is a useful command for getting information from DNS server. It is a network administration tool for querying the Domain Name System (DNS) to obtain domain name or IP address mapping or any other specific DNS record. It is also used to troubleshoot DNS related problems.

The **nslookup** command queries internet domain name servers in two modes. Interactive mode allows you to query name servers for information about various hosts and domains, or to print a list of the hosts in a domain. In noninteractive mode, the names and requested information are printed for a specified host or domain.

```
C:\WINDOWS\system32>nslookup google.com
Server:  UnKnown
Address:  192.168.0.1

Non-authoritative answer:
Name:    google.com
Addresses:  2404:6800:4009:82f::200e
          216.58.203.14
```

Lookup for an soa record
SOA record (start of authority), provides the authoritative information about the domain, the e-mail address of the domain admin, the domain serial number, etc…

```
C:\WINDOWS\system32>nslookup -type=soa google.com
Server:  UnKnown
Address:  192.168.0.1

Non-authoritative answer:
google.com
        primary name server = ns1.google.com
        responsible mail addr = dns-admin.google.com
        serial  = 447762907
        refresh = 900 (15 mins)
        retry   = 900 (15 mins)
        expire  = 1800 (30 mins)
        default TTL = 60 (1 min)
```

Lookup for an ns record
NS (Name Server) record maps a domain name to a list of DNS servers authoritative for that domain. It will output the name serves which are associated with the given domain.

```
C:\WINDOWS\system32>nslookup -type=ns google.com
Server:  UnKnown
Address:  192.168.0.1

Non-authoritative answer:
google.com      nameserver = ns4.google.com
google.com      nameserver = ns1.google.com
google.com      nameserver = ns2.google.com
google.com      nameserver = ns3.google.com
```

traceroute:

The **traceroute** command is intended for use in network testing, measurement, and management. It prints the route that IP packets take to a network host. Traceroute is the route tracing tool used on Unix-like Operating Systems (including Mac OS X).
traceroute command in Linux prints the route that a packet takes to reach the host. This command is useful when you want to know about the route and about all the hops that a packet takes. The first column corresponds to the hop count. The second column represents the address of that hop and after that, you see three space-separated time in milliseconds. *traceroute* command sends three packets to the hop and each of the time refers to the time taken by the packet to reach the hop.

The Windows tracert tool determines the route to a destination by sending ICMP packets to the destination.
In these packets, tracert uses varying IP Time-To-Live (TTL) values.
The TTL is effectively a hop counter, where a hop is a location that the packet stops at, to reach the destination.
The tool may take some time to complete (particularly if there is a problem), as the tool waits for responses (which may not come).

```
C:\WINDOWS\system32>tracert google.com

Tracing route to google.com [142.250.183.174]
over a maximum of 30 hops:

  1    15 ms     6 ms     4 ms  ARCHER_C5 [192.168.0.1]
  2     3 ms     7 ms     3 ms  host-58125228.fivenetwork.com [58.146.125.128]
  3     8 ms     3 ms     9 ms  10.20.99.2
  4     3 ms    16 ms     8 ms  10.20.31.210
  5     9 ms     3 ms     9 ms  72.14.197.213
  6    16 ms     6 ms     4 ms  108.170.248.193
  7     3 ms     7 ms     3 ms  142.251.64.13
  8     4 ms     7 ms    32 ms  bom07s32-in-f14.1e100.net [142.250.183.174]

Trace complete.
```

Conclusion:

Thus, we have successfully implemented and studied the use of network reconnaissance tools like WHOIS, dig, traceroute, nslookup to gather information about networks and domain registrars.

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Experiment 8

Date of Performance : 12-05-2022 **Date of Submission:** 03-06-2022

SAP Id: 60004200059 **Name : Vaishnavi Sawant**
Div: A **Batch : A2**

Aim of Experiment

Study of packet sniffertools :wireshark, :

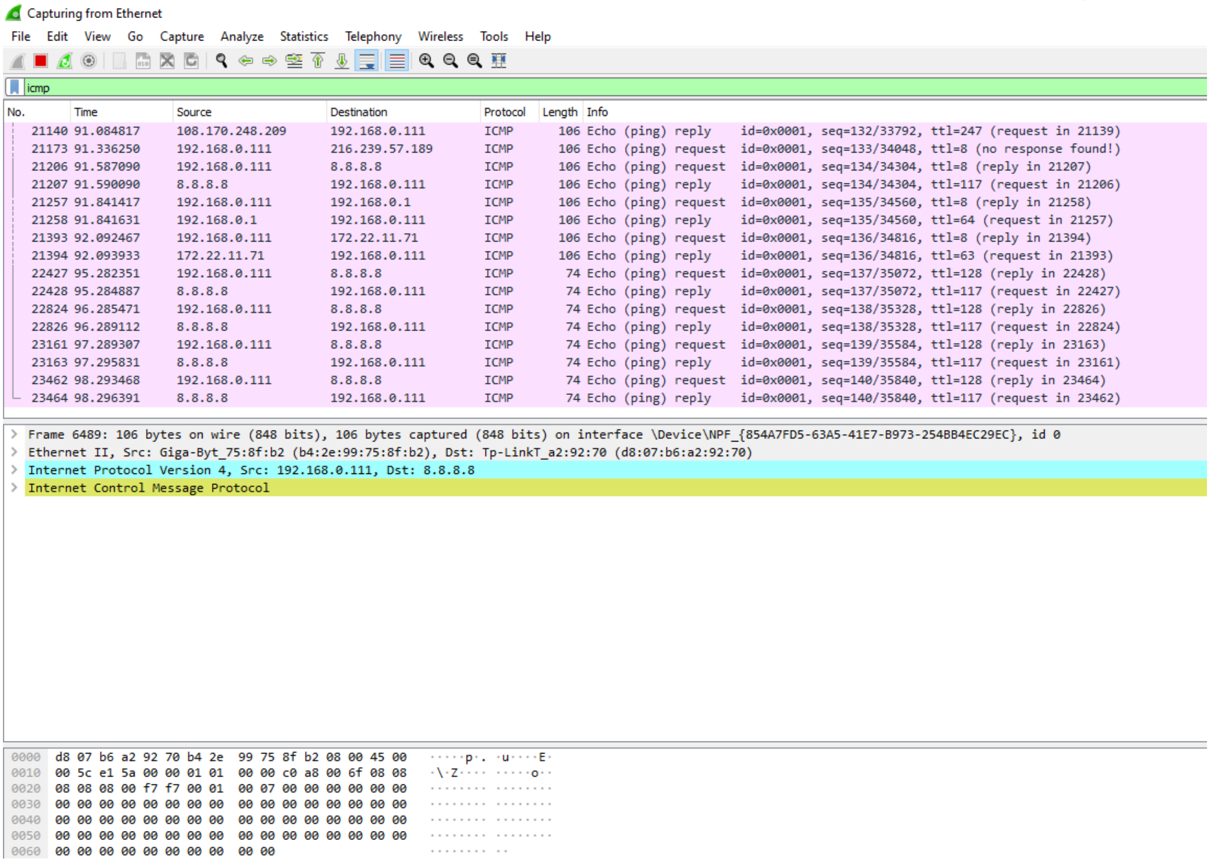
a) Download and installwiresharkand captureicmp,tcp, and http packets in promiscuous mode
b) Explore how the packets can be traced based on different filters. (CO5)

Theory:

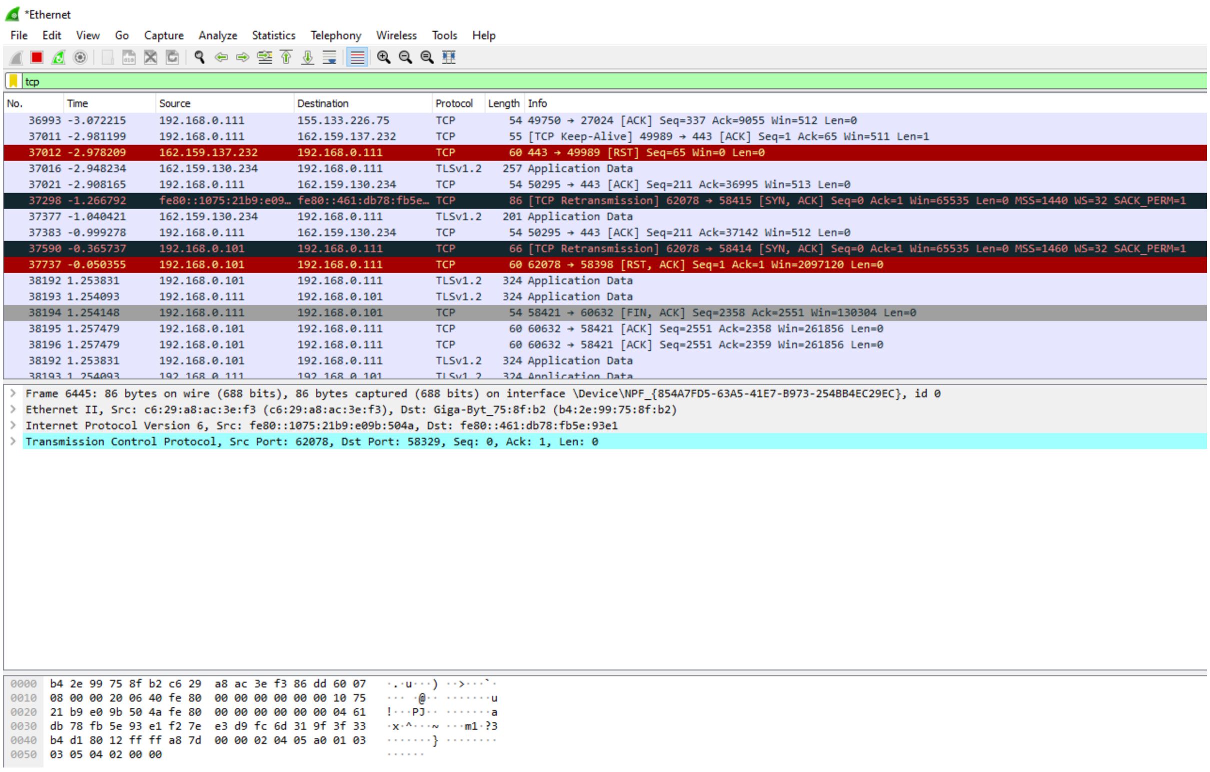
Wireshark is a freeand open-sourcepacket analyzer. It is used for networktroubleshooting, analysis, software and communications protocoldevelopment, and education.Wireshark lets the user put network interface controllers into promiscuous mode (if supported by the network interface controller), so they can see all the traffic visible on that interface including unicast traffic not sent to that network interface controller's MAC address. However, when capturing with a packet analyzer in promiscuous mode on a port on a network switch, not all traffic through the switch is necessarily sent to the port where the capture is done, so capturing in promiscuous mode is not necessarily sufficient to see all network traffic. Port mirroring or various network taps extend capture to any point on the network. Simple passive taps are extremely resistant to tampering.

Capturing ICMP Packets:

```
F:\>ping 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=2ms TTL=117
Reply from 8.8.8.8: bytes=32 time=3ms TTL=117
Reply from 8.8.8.8: bytes=32 time=6ms TTL=117
Reply from 8.8.8.8: bytes=32 time=3ms TTL=117

Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 6ms, Average = 3ms
```



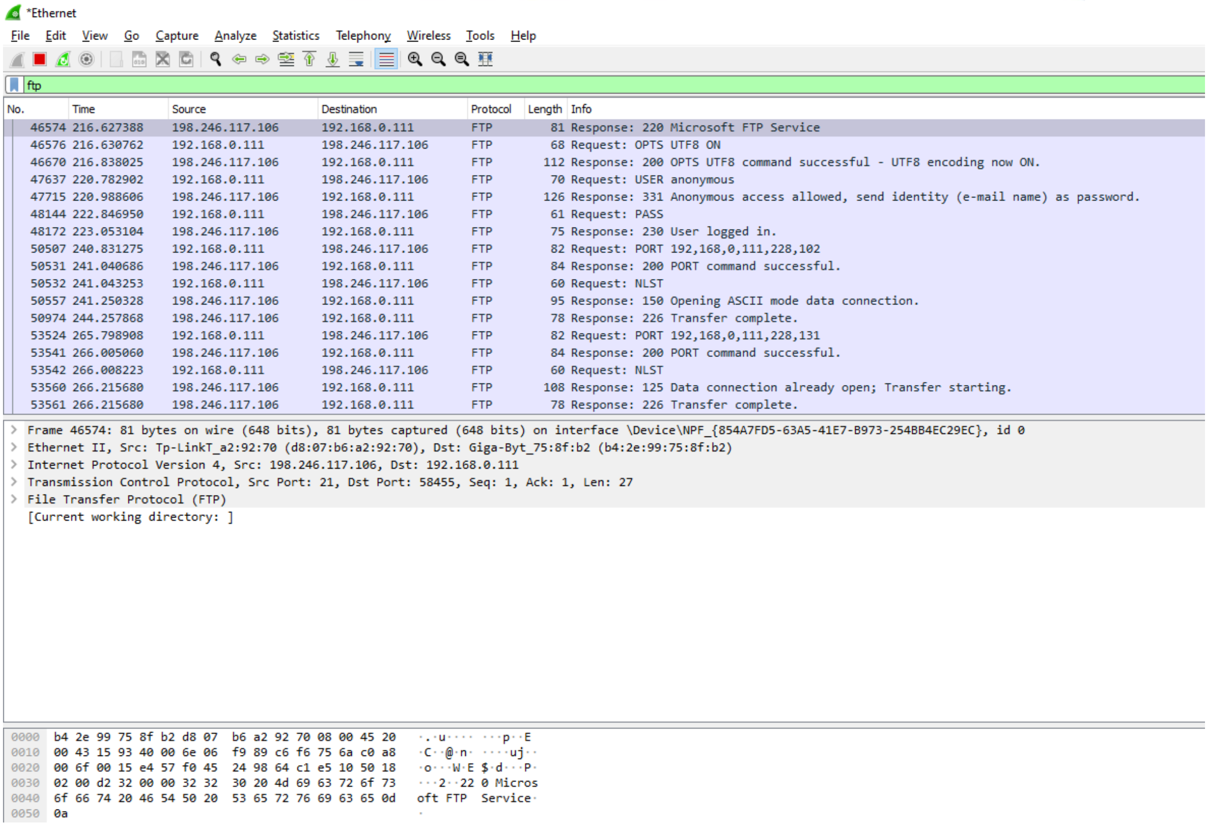
Capturing TCP Packets:



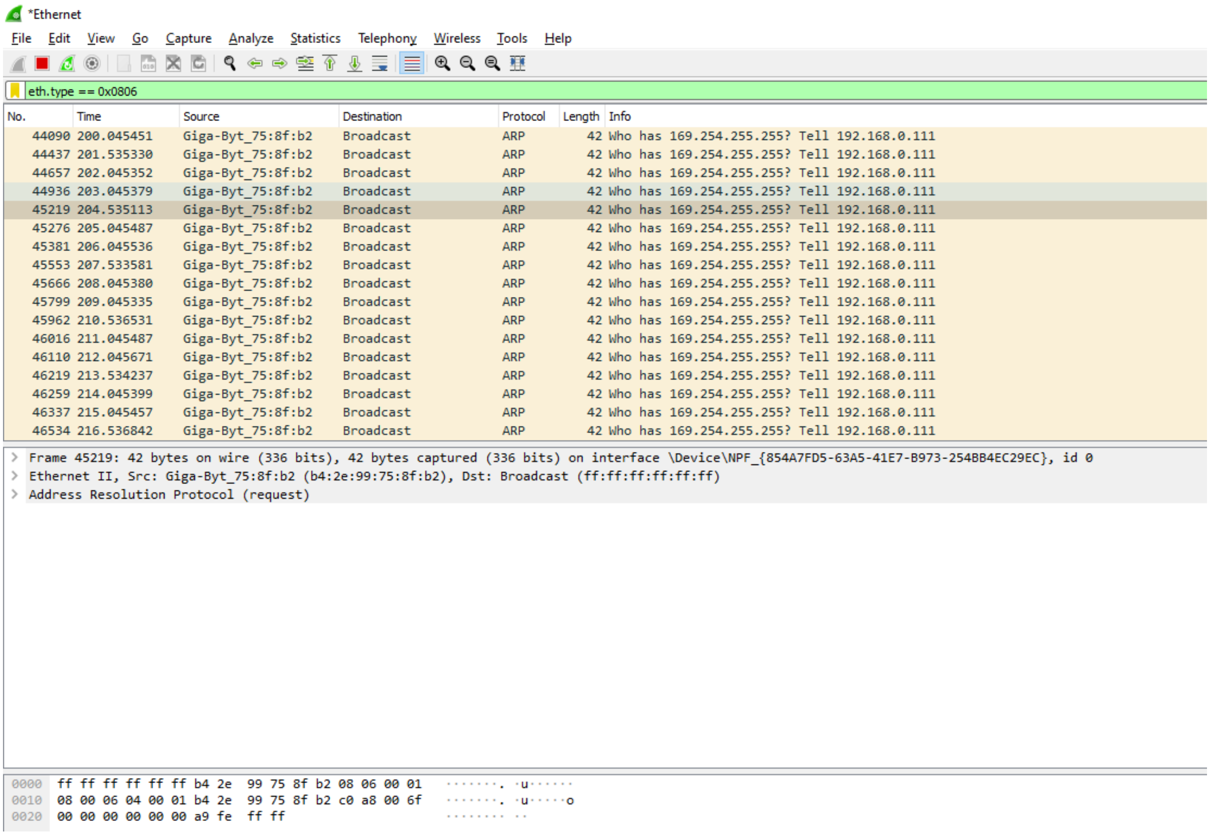
Capturing FTP packets:

C:\Windows\System32\cmd.exe - ftp ftp.cdc.gov

```
F:\>ftp ftp.cdc.gov
Connected to ftp.cdc.gov.
220 Microsoft FTP Service
200 OPTS UTF8 command successful - UTF8 encoding now ON.
User (ftp.cdc.gov:(none)): anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
Password:
230 User logged in.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection.
.change.dir
.message
pub
Readme
Siteinfo
w3c
welcome.msg
226 Transfer complete.
ftp: 67 bytes received in 0.00Seconds 16.75Kbytes/sec.
ftp> ls
200 PORT command successful.
125 Data connection already open; Transfer starting.
.change.dir
.message
pub
Readme
Siteinfo
w3c
welcome.msg
226 Transfer complete.
ftp: 67 bytes received in 0.01Seconds 9.57Kbytes/sec.
ftp>
```



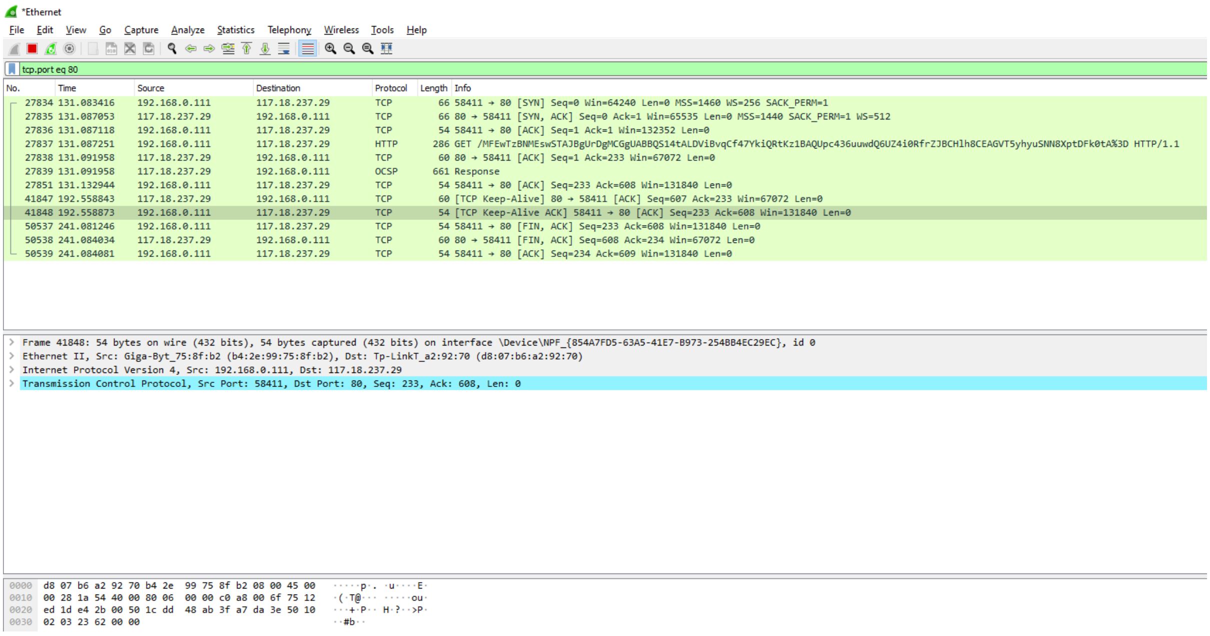
Capturing ARP Packets:



B] Tracing Packets based on filters:

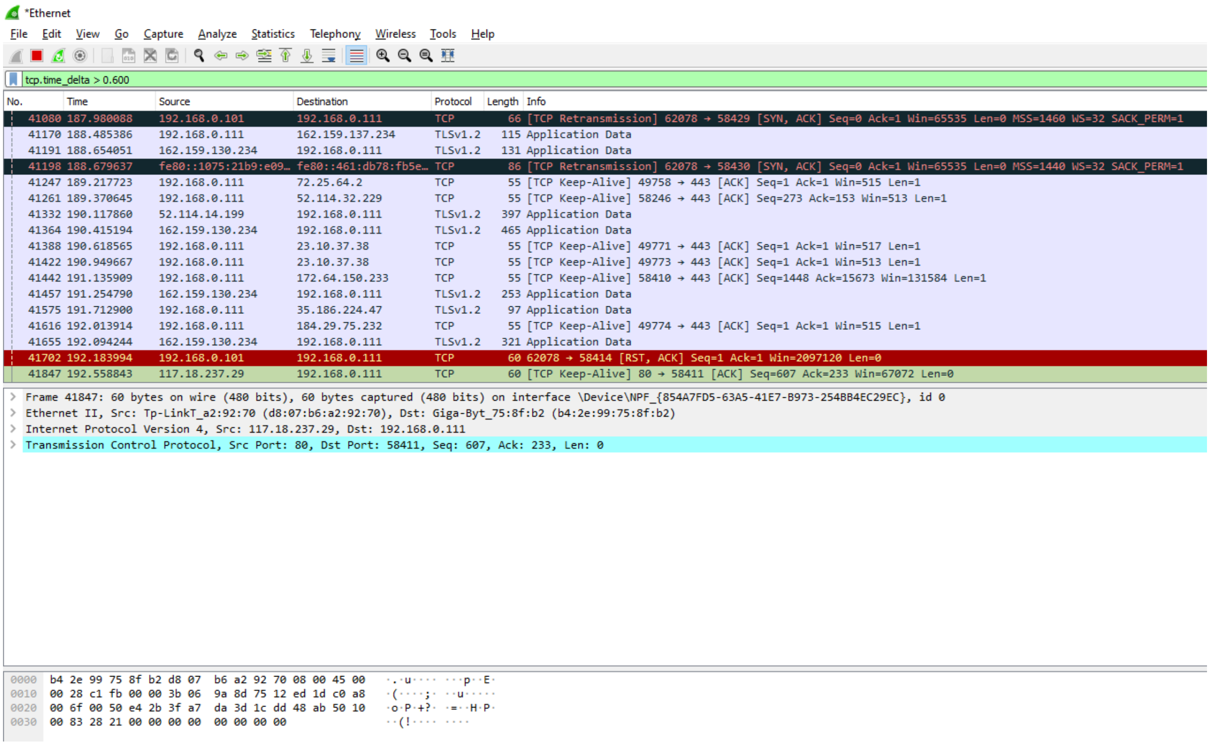
1] Filter Results by Port:

Traces all packets related to Port 80



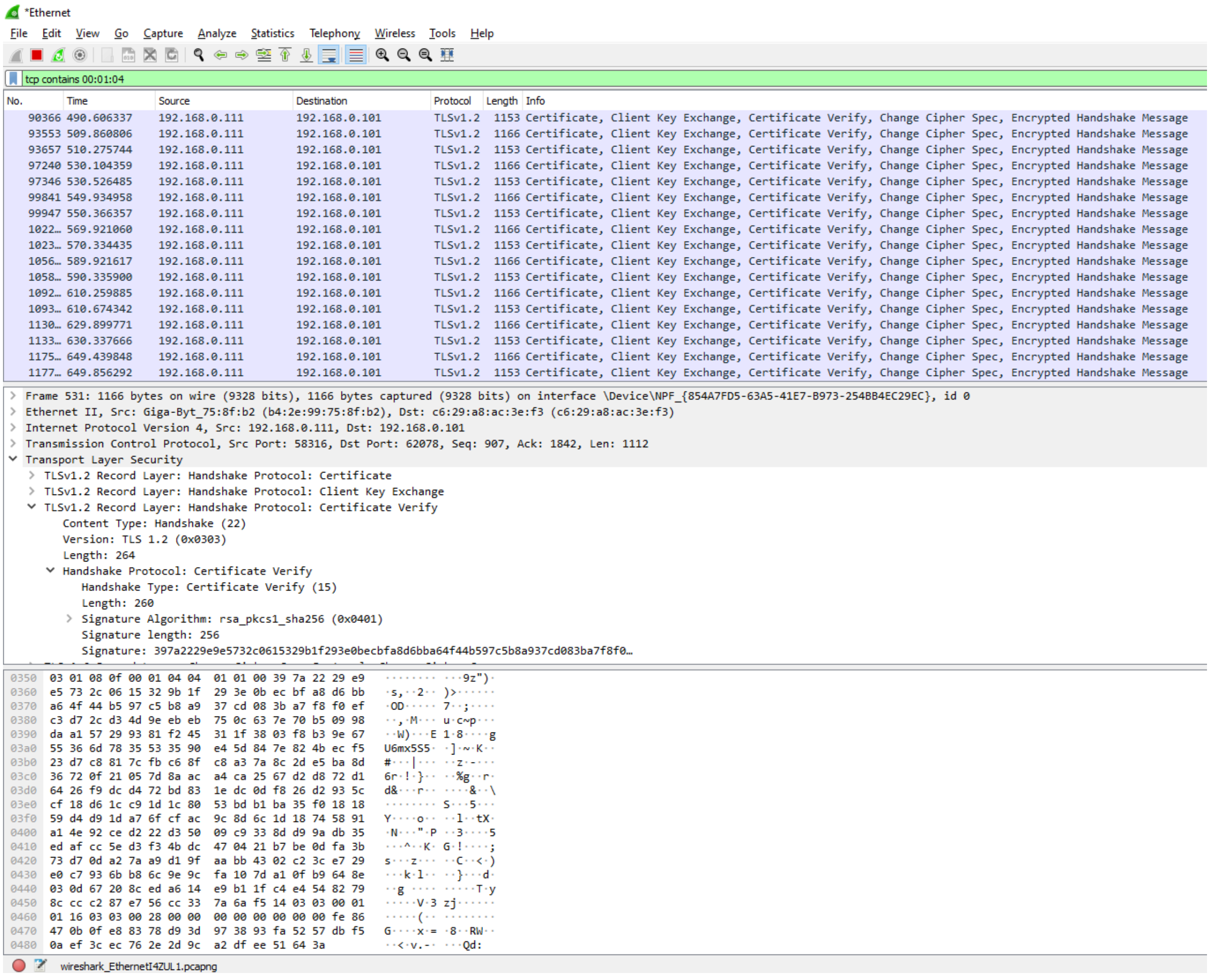
2] Filter by Delta Time:

Displays tcp packets with delta time of greater than 0.600 sec



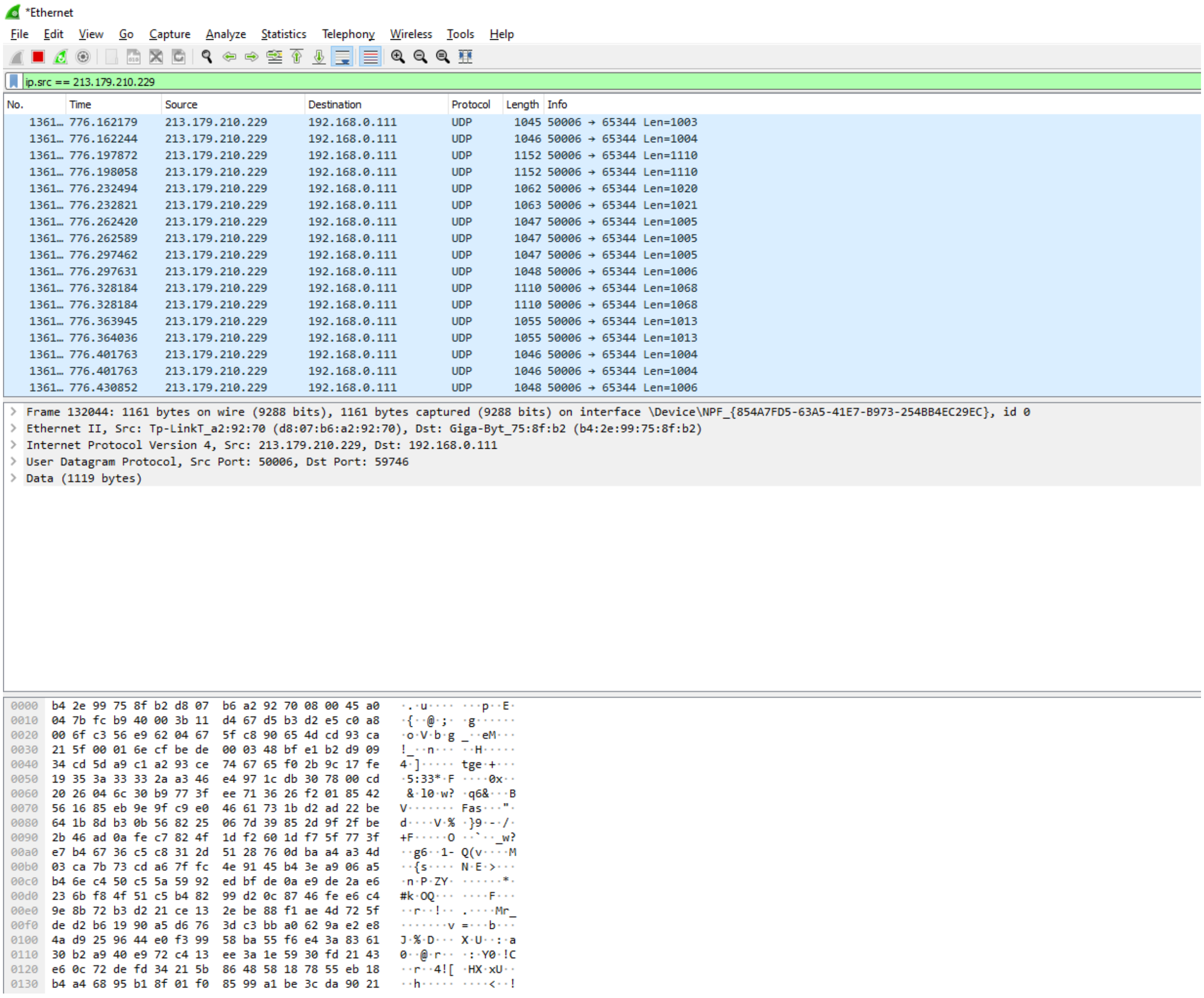
3] Filter by Byte Sequence:

Displays packets which contain a particular byte sequence.



4] Filter by Source IP Address:

Displays packets which have source IP address same as the one provided in the argument



Conclusion:

Thus, we have successfully studied packet sniffing tools (Wireshark) and explored how packets can be traced on basis of different filters.

Experiment 9

Date of Performance : 19-05-2022 **Date of Submission: 03-06-2022**

SAP Id: 60004200059 **Name : Vaishnavi Sawant**
Div: A **Batch : A2**

Aim of Experiment

Implementation of Network Intrusion Detection System using NMAP, SNORT and IPTABLE (CO6).

Theory:

IPTables:

iptables is a user-space utility program that allows a system administrator to configure the IP packet filter rules of the Linux kernel firewall, implemented as different Netfilter modules. The filters are organized in different tables, which contain chains of rules for how to treat network traffic packets. Different kernel modules and programs are currently used for different protocols; iptables applies to IPv4, ip6tables to IPv6, arptables to ARP, and ebtables to Ethernet frames.

iptables requires elevated privileges to operate and must be executed by user root, otherwise it fails to function. On most Linux systems, iptables is installed as /usr/sbin/iptables and documented in its man pages, which can be opened using man iptables when installed. It may also be found in /sbin/iptables, but since iptables is more like a service rather than an "essential binary", the preferred location remains /usr/sbin.

NMAP:

Nmap is a network mapper that has emerged as one of the most popular, free network discovery tools on the market. Nmap is now one of the core tools used by network administrators to map their networks. The program can be used to find live hosts on a network, perform port scanning, ping sweeps, OS detection, and version detection.

```
c:\>nmap 10.130.65.29 -O -sV -p 20-25 -Pn
Starting Nmap 7.92 ( https://nmap.org ) at 2022-05-12 22:08 India Standard Time
Nmap scan report for 10.130.65.29
Host is up.

PORT   STATE    SERVICE   VERSION
20/tcp filtered ftp-data
21/tcp filtered ftp
22/tcp filtered ssh
23/tcp filtered telnet
24/tcp filtered priv-mail
25/tcp filtered smtp
Too many fingerprints match this host to give specific OS details

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 29.76 seconds
```

```
c:\>nmap 10.130.65.29 10.130.65.28 -sL
Starting Nmap 7.92 ( https://nmap.org ) at 2022-05-12 22:09 India Standard Time
Nmap scan report for 10.130.65.29
Nmap scan report for 10.130.65.28
Nmap done: 2 IP addresses (0 hosts up) scanned in 0.06 seconds
```

```
c:\>nmap 10.130.65.29 -p 21,22,23,25,80 -Pn
Starting Nmap 7.92 ( https://nmap.org ) at 2022-05-12 22:10 India Standard Time
Nmap scan report for 10.130.65.29
Host is up (0.0050s latency).

PORT   STATE    SERVICE
21/tcp filtered ftp
22/tcp filtered ssh
23/tcp filtered telnet
25/tcp filtered smtp
80/tcp filtered http

Nmap done: 1 IP address (1 host up) scanned in 2.32 seconds
```

SNORT:

Snort is a free and open-source network intrusion prevention and detection system.

It uses a rule-based language combining signature, protocol, and anomaly inspection methods to detect malicious activity such as denial-of-service (DoS) attacks, Buffer overflows, stealth port scans, CGI attacks, SMB probes, and OS fingerprinting attempts.

It is capable of performing real-time traffic analysis and packet logging on IP networks.

IP Protocols supported by SNORT:

As we know, IP is a unique address for every computer and is used for transferring data or packets over the internet from one network to the other network. Each packet contains a message, data, source, destination address, and much more. Snort supports three IP protocols for suspicious behavior:

- Transmission Control Protocol (TCP) Connects two different hosts and exchanges data between them. Examples include HTTP, SMTP, and FTP.

- User Datagram Protocol (UDP): Broadcasts messages over the internet. Examples include DNS traffic.
- Internet Control Message Protocol (ICMP): Sends network error messages in Windows. Examples include Ping and Traceroute.

Snort Rules:

Rules are a different methodology for performing detection, which bring the advantage of 0-day detection to the table.

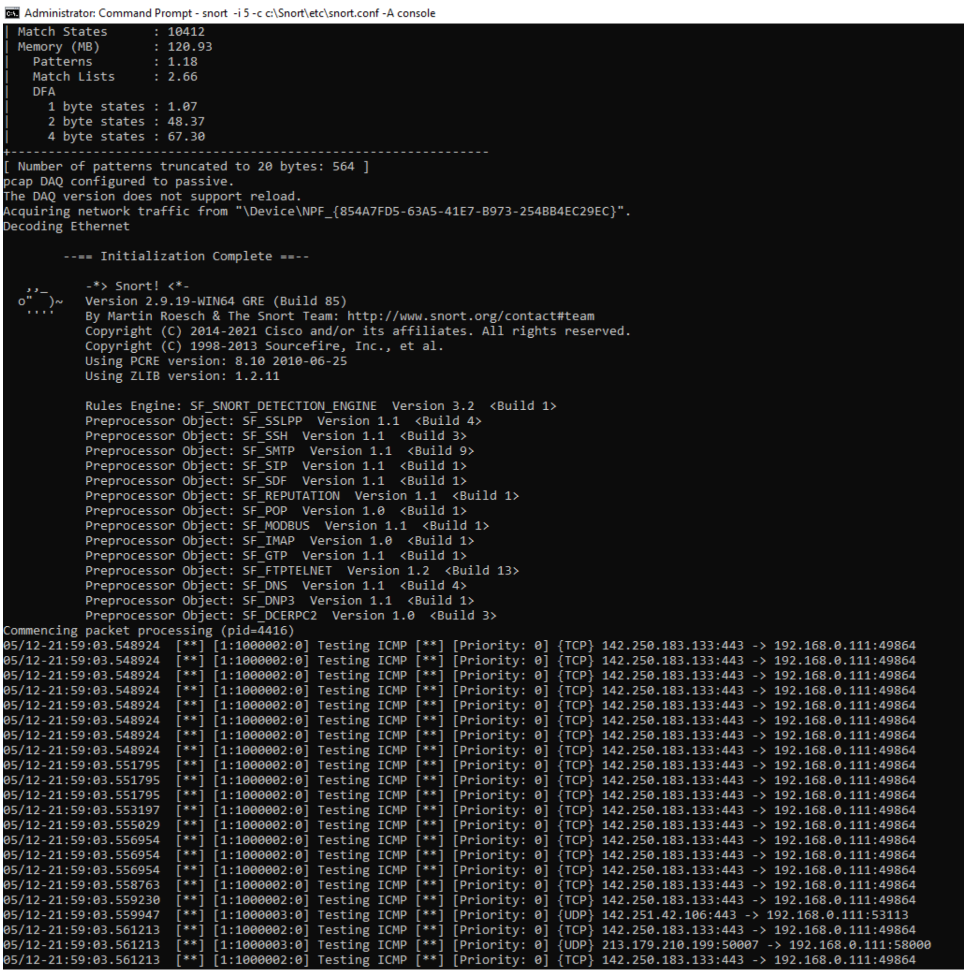
Developing a rule requires an acute understanding of how the vulnerability actually works.

Snort generates alerts according to the rules defined in the configuration file.

The Snort rule language is very flexible, and creation of new rules is relatively simple.

Snort rules help in differentiating between normal internet activities and malicious activities

ICMP Intrusion Detection:



```
Administrator: Command Prompt - snort  -i 5 -c c:\Snort\etc\snort.conf -A console
| Match States      : 10412
| Memory (MB)       : 120.93
|   Patterns        : 1.18
|   Match Lists     : 2.66
|   DFA
|     1 byte states : 1.07
|     2 byte states : 48.37
|     4 byte states : 67.30
+----------------------------------------------------------------
[ Number of patterns truncated to 20 bytes: 564 ]
pcap DAQ configured to passive.
The DAQ version does not support reload.
Acquiring network traffic from "\Device\NPF_{854A7FD5-63A5-41E7-B973-254BB4EC29EC}".
Decoding Ethernet

        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.19-WIN64 GRE (Build 85)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2021 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using PCRE version: 8.10 2010-06-25
           Using ZLIB version: 1.2.11

           Rules Engine: SF_SNORT_DETECTION_ENGINE  Version 3.2  <Build 1>
           Preprocessor Object: SF_SSLPP  Version 1.1  <Build 4>
           Preprocessor Object: SF_SSH  Version 1.1  <Build 3>
           Preprocessor Object: SF_SMTP  Version 1.1  <Build 9>
           Preprocessor Object: SF_SIP  Version 1.1  <Build 1>
           Preprocessor Object: SF_SDF  Version 1.1  <Build 1>
           Preprocessor Object: SF_REPUTATION  Version 1.1  <Build 1>
           Preprocessor Object: SF_POP  Version 1.0  <Build 1>
           Preprocessor Object: SF_MODBUS  Version 1.1  <Build 1>
           Preprocessor Object: SF_IMAP  Version 1.0  <Build 1>
           Preprocessor Object: SF_GTP  Version 1.1  <Build 1>
           Preprocessor Object: SF_FTPTELNET  Version 1.2  <Build 13>
           Preprocessor Object: SF_DNS  Version 1.1  <Build 4>
           Preprocessor Object: SF_DNP3  Version 1.1  <Build 1>
           Preprocessor Object: SF_DCERPC2  Version 1.0  <Build 3>
Commencing packet processing (pid=4416)
05/12-21:59:03.548924  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.548924  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.548924  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.548924  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.548924  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.548924  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.548924  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.548924  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.551795  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.551795  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.551795  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.553197  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.555029  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.556954  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.556954  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.556954  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.558763  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.559230  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.559947  [**] [1:1000003:0] Testing ICMP [**] [Priority: 0] {UDP} 142.251.42.106:443 -> 192.168.0.111:53113
05/12-21:59:03.561213  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
05/12-21:59:03.561213  [**] [1:1000003:0] Testing ICMP [**] [Priority: 0] {UDP} 213.179.210.199:50007 -> 192.168.0.111:58000
05/12-21:59:03.561213  [**] [1:1000002:0] Testing ICMP [**] [Priority: 0] {TCP} 142.250.183.133:443 -> 192.168.0.111:49864
```

Conclusion:

Thus, we have successfully implemented Network Intrusion Detection System usingNMAP,SNORTand IPTables.

Experiment 10

Date of Performance : 26-05-2022 **Date of Submission: 03-06-2022**

SAP Id: 60004200059 **Name : Vaishnavi Sawant**
Div: A **Batch : A2**

Aim of Experiment

Implement Buffer Overflow Attack. (CO7)

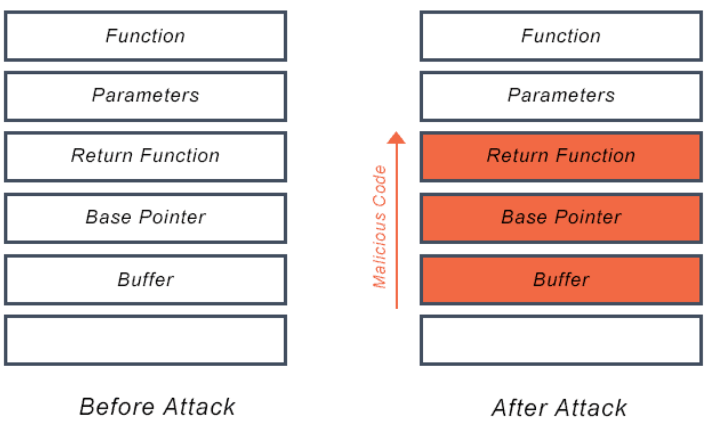
Theory:

Buffer Overflow Attack:

A buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers. A buffer overflow, or buffer overrun, occurs when more data is put into a fixed-length buffer than the buffer can handle. The extra information, which has to go somewhere, can overflow into adjacent memory space, corrupting or overwriting the data held in that space. This overflow usually results in a system crash, but it also creates the opportunity for an attacker to run arbitrary code or manipulate the coding errors to prompt malicious actions.

Many programming languages are prone to buffer overflow attacks. However, the extent of such attacks varies depending on the language used to write the vulnerable program. For instance, code written in Perl and JavaScript is generally not susceptible to buffer overflows. However, a buffer overflow in a program written in C, C++, Fortran or Assembly could allow the attacker to fully compromise the targeted system.

Buffer Overflow Attack



A buffer overflow vulnerability will typically occur when code:

1. Is reliant on external data to control its behavior
2. Is dependent on data properties that are enforced beyond its immediate scope
3. Is so complex that programmers are not able to predict its behavior accurately

Buffer Overflow Consequences

Common consequences of a buffer overflow attack include the following:

1. **System crashes:** A buffer overflow attack will typically lead to the system crashing. It may also result in a lack of availability and programs being put into an infinite loop.
2. **Access control loss:** A buffer overflow attack will often involve the use of arbitrary code, which is often outside the scope of programs' security policies.
3. **Further security issues:** When a buffer overflow attack results in arbitrary code execution, the attacker may use it to exploit other vulnerabilities and subvert other security services.

Types of Buffer Overflow Attacks

There are several types of buffer overflow attacks that attackers use to exploit organizations' systems. The most common are:

1. **Stack-based buffer overflows:** This is the most common form of buffer overflow attack. The stack-based approach occurs when an attacker sends data containing malicious code to an application, which stores the data in a stack buffer. This overwrites the data on the stack, including its return pointer, which hands control of transfers to the attacker.
2. **Heap-based buffer overflows:** A heap-based attack is more difficult to carry out than the stack-based approach. It involves the attack flooding a program's memory space beyond the memory it uses for current runtime operations.
3. **Format string attack:** A format string exploit takes place when an application processes input data as a command or does not validate input data effectively. This enables the attacker to execute code, read data in the stack, or cause segmentation faults in the application. This could trigger new actions that threaten the security and stability of the system.

Ollydbg:

OllyDbg is a 32-bit debugging tool used to analyze binary code. Its popularity is tied to the fact that people can do so despite not having access to the source code. OllyDbg can be used to evaluate and debug malware. OllyDbg is a popular debugger due to its ease of use and being freeware.

OllyDbg is useful in analyzing malware. If you plan to analyze malware on your own, you want to ensure you have your environment setup to protect yourself and your assets. This should be done in a closed environment within a virtual machine. Using a virtual machine is not enough. Do some research on best ways to isolate your environment. Avoid using bridged mode, as it leaves your network exposed.

Splint:

Splint is a tool for statically checking C programs for security vulnerabilities and programming mistakes. Splint does many of the traditional lint checks including unused declarations, type inconsistencies, use before definition, unreachable code, ignored return values, execution paths with no return, likely infinite loops, and fall through cases. More powerful checks are made possible by additional information given in source code annotations. Annotations are stylized comments that document assumptions about functions, variables, parameters and types. In addition to the checks specifically enabled by annotations, many of the traditional lint checks are improved by exploiting this additional information.

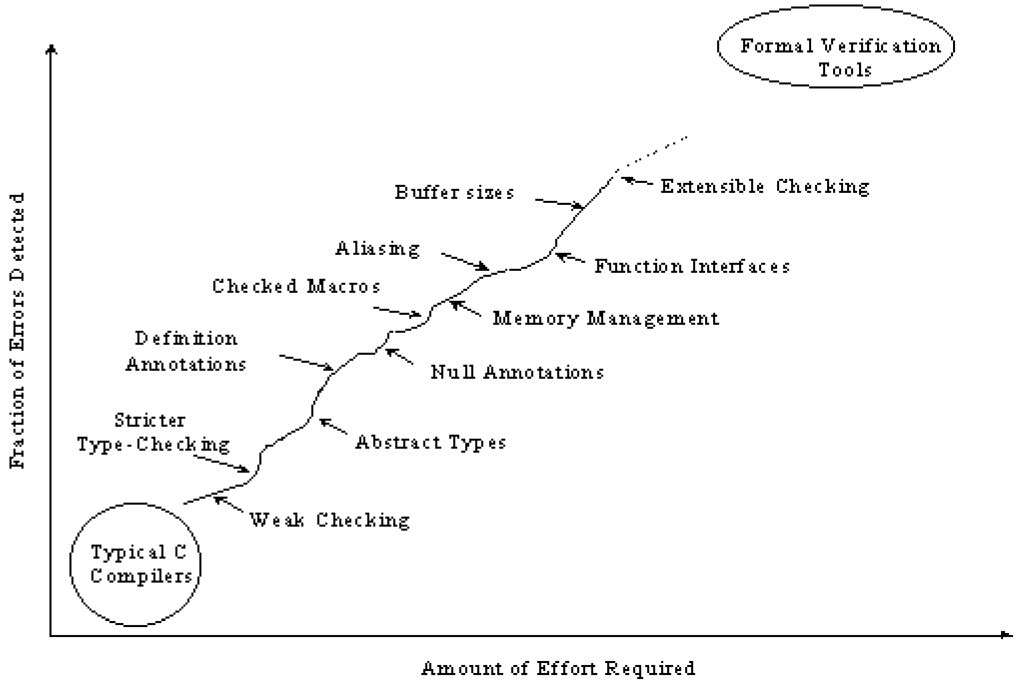


Figure 1. Typical Effort-Benefit Curve

As more effort is put into annotating programs, better checking results. A representational effort-benefit curve for using Splint is shown in Figure 1. Splint is designed to be flexible and allow programmers to select appropriate points on the effort-benefit curve for particular projects. As different checks are turned on and more information is given in code annotations the number of bugs that can be detected increases dramatically.

Problems detected by Splint include:

1. Dereferencing a possibly null pointer.
2. Using possibly undefined storage or returning storage that is not properly defined.
3. Type mismatches, with greater precision and flexibility than provided by C compilers.
4. Violations of information hiding.
5. Memory management errors including uses of dangling references and memory leaks.
6. Dangerous aliasing.
7. Modifications and global variable uses that are inconsistent with specified interfaces.
8. Problematic control flow such as likely infinite loops, fall through cases or incomplete switches, and suspicious statements.
9. Buffer overflow vulnerabilities
10. Dangerous macro implementations or invocations
11. Violations of customized naming conventions.

Cppcheck:

Cppcheck is a command-line tool that tries to detect bugs that your C/C++ compiler doesn't see. It is versatile, and can check non-standard code including various compiler extensions, inline assembly code, etc. Its internal preprocessor can handle includes, macros, and several preprocessor commands. While Cppcheck is highly configurable, you can start using it just by giving it a path to the source code.

Cppcheck supports a wide variety of static checks that may not be covered by the compiler itself. These checks are static analysis checks that can be performed at a source code level. The program is directed towards static analysis checks that are rigorous, rather than heuristic in nature.

Some of the checks that are supported include:

- Automatic variable checking
- Bounds checking for array overruns
- Classes checking (e.g. unused functions, variable initialization and memory duplication)

Program:

Code used to show Buffer Overflow

```
#include <stdio.h>
#include <string.h>
#define UP_MAXLEN 20
#define UP_PAIR_COUNT 3
int main() {
 int flag;
 char termBuf;
 char username[UP_MAXLEN];
 char cpass[UP_MAXLEN];
 char npass[UP_MAXLEN];
 char keys[UP_PAIR_COUNT][2][UP_MAXLEN] = {
 {"Admin", "pass3693"},
 {"Ahmed", "slimshady1234"},
 {"Vaishnavi","vaish123"}
};
 while (1)
 {
 flag = 0;
 printf("Change Password\n");
 printf("Enter Username: "); gets(username);
 printf("Enter Current Password: "); gets(cpass);
 for(int i = 0; i < UP_PAIR_COUNT; i++) {
 if (strcmp(keys[i][0], username) == 0 && strcmp(keys[i][1], cpass) == 0) {
 printf("Enter New Password: "); gets(npass);
 strcpy(&keys[i][1][0], npass);
 for(int j = 0; j < UP_PAIR_COUNT; j++) printf("%s | %s\n", keys[j][0], keys[j][1]
);
 printf("Password Changed!\n");
 printf("Continue? Y/N: ");
 gets(&termBuf);
 if (termBuf != 'Y') return 0;
 else flag = 1;
 }
 }
 if (flag == 1) continue;
 printf("Incorrect Username and Password. Enter Y to continue.\n");
 gets(&termBuf);
 if (termBuf != 'Y') return 0;
 }
}
```

<u>Output</u>

Buffer Overflow Attack

```
Change Password
Enter Username: Admin
Enter Current Password: pass3693
Enter New Password: mnbvcxzlkjhgfdsapoiuytrewqCRYPTOGRAPHY
Admin | mnbvcxzlkjhgfdsapoiuytrewqCRYPTOGRAPHY
ytrewqCRYPTOGRAPHY | slimshady1234
Vaishnavi | vaish123
Password Changed!
Continue? Y/N: Y
Change Password
Enter Username: ytrewqCRYPTOGRAPHY
Enter Current Password: slimshady1234
Enter New Password: ahmed123
Admin | mnbvcxzlkjhgfdsapoiuytrewqCRYPTOGRAPHY
ytrewqCRYPTOGRAPHY | ahmed123
Vaishnavi | vaish123
Password Changed!
Continue? Y/N: Y
Change Password
Enter Username: Vaishnavi
Enter Current Password: vaish123
Enter New Password: qwertyuiopzxcvbnmasdfghjklCryptography
Admin | mnbvcxzlkjhgfdsapoiuytrewqCRYPTOGRAPHY
ytrewqCRYPTOGRAPHY | ahmed123
Vaishnavi | qwertyuiopzxcvbnmasdfghjklCryptography
Password Changed!
Continue? Y/N: N
*** stack smashing detected ***: terminated
Aborted
```

Code after fixing the Buffer Overflow Vulnerability

```
#include <stdio.h>
#include <string.h>
#define UP_MAXLEN 20
#define UP_PAIR_COUNT 3
int main() {
int flag;
char termBuf;
char username[UP_MAXLEN];
char cpass[UP_MAXLEN];
char npass[UP_MAXLEN];
char keys[UP_PAIR_COUNT][2][UP_MAXLEN] = {
{"Admin", "pass3693"},
{"Ahmed", "slimshady1234"},
{"Vaishnavi","vaish123"}
};
while (1)
{
flag = 0;
printf("Change Password\n");
printf("Enter Username: ");
fgets(username, UP_MAXLEN, stdin);
username[strcspn(username, "\r\n")] = 0;
printf("Enter Current Password: ");
fgets(cpass, UP_MAXLEN, stdin);
cpass[strcspn(cpass, "\r\n")] = 0;
for(int i = 0; i< UP_PAIR_COUNT; i++) {
if (strcmp(keys[i][0], username) == 0 && strcmp(keys[i][1], cpass) == 0) {
printf("Enter New Password: ");
fgets(npass, UP_MAXLEN, stdin);
npass[strcspn(npass, “\n")] = 0;
strcpy(&keys[i][1][0], npass);
for(int j = 0; j < UP_PAIR_COUNT; j++) printf("%s | %s\n", keys[j][0], keys[j][1]);
printf("Password Changed!\n");
printf("Continue? Y/N: ");
scanf("%c", &termBuf);
if (termBuf != "Y") return 0;
else flag = 1;
while((termBuf = getchar()) != "\n" && termBuf != EOF);
}
}
if (flag == 1) continue;
printf("Incorrect Username and Password. Enter Y to continue.\n");
scanf("%c", &termBuf);
if (termBuf != 'Y') return 0;
while((termBuf = getchar()) != "\n" && termBuf != EOF);
```

```
}
}
```

Output:

Buffer Overflow Attack not working on the fixed Code

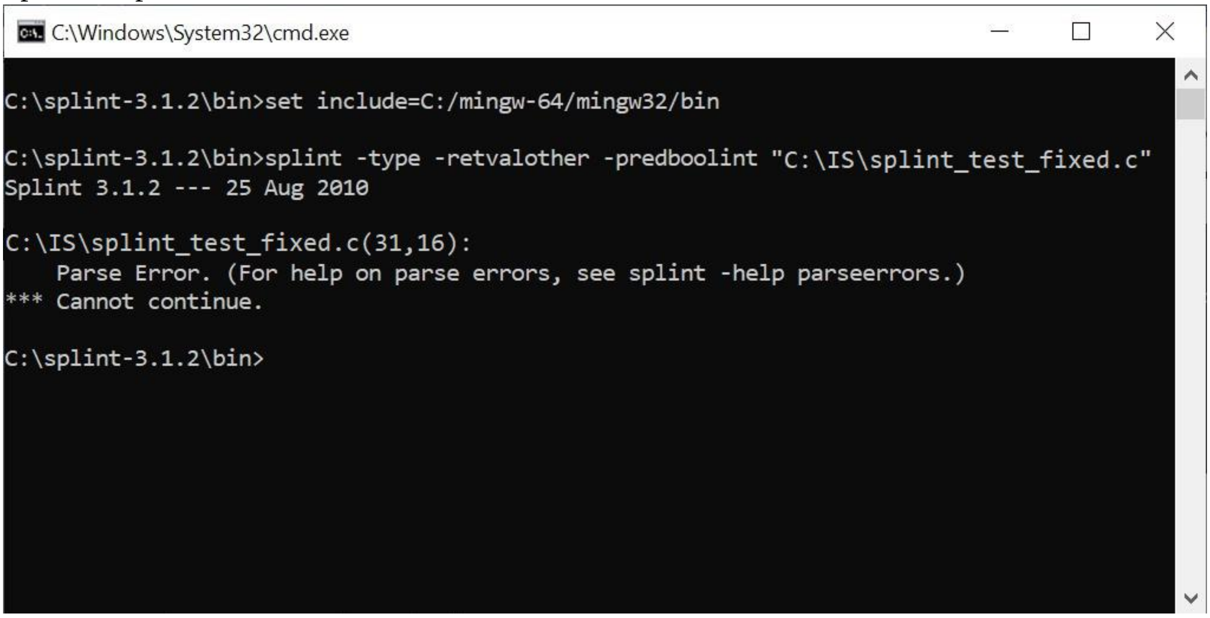
```
Change Password
Enter Username: Ahmed
Enter Current Password: slimshady1234
Enter New Password: mnbvcxzlkjhgfdsapoiuytrewqcrypto
Admin | pass3693
Ahmed | mnbvcxzlkjhgfdsapoi
Vaishnavi | vaish123
Password Changed!
Continue? Y/N: Y
```

```
Change Password
Enter Username: Vaishnavi
Enter Current Password: vaish123
Enter New Password: asdfghjklqwertyuiopzxcvbnmcrypto
Admin | pass3693
Ahmed | slimshady1234
Vaishnavi | asdfghjklqwertyuiop
Password Changed!
Continue? Y/N: N
```

Splint Output for the Vulnerable Code

```
C:\Windows\System32\cmd.exe
C:\splint-3.1.2\bin>set include=C:/mingw-64/mingw32/bin

C:\splint-3.1.2\bin>splint -type -retvalother -predboolint "C:\IS\splint_test_fixed.c"
Splint 3.1.2 --- 25 Aug 2010

C:\IS\splint_test_fixed.c: (in function main)
C:\IS\splint_test_fixed.c(22,37):
    Use of gets leads to a buffer overflow vulnerability.  Use fgets instead:
    gets
  Use of function that may lead to buffer overflow. (Use -bufferoverflowhigh to
  inhibit warning)
C:\IS\splint_test_fixed.c(23,45):
    Use of gets leads to a buffer overflow vulnerability.  Use fgets instead:
    gets
C:\IS\splint_test_fixed.c(24,16):
    Parse Error. (For help on parse errors, see splint -help parseerrors.)
*** Cannot continue.

C:\splint-3.1.2\bin>
```

Splint Output for the Fixed Code



```
C:\Windows\System32\cmd.exe

C:\splint-3.1.2\bin>set include=C:/mingw-64/mingw32/bin

C:\splint-3.1.2\bin>splint -type -retvalother -predboolint "C:\IS\splint_test_fixed.c"
Splint 3.1.2 --- 25 Aug 2010

C:\IS\splint_test_fixed.c(31,16):
    Parse Error. (For help on parse errors, see splint -help parseerrors.)
*** Cannot continue.

C:\splint-3.1.2\bin>
```

Conclusion:

Thus, Buffer Overflow Attack has been successfully demonstrated and prevented using Splint programming tool.