# DON BOSCO INSTITUTE OF TECHNOLOGY

EXPERIMENT – 1

ITL502.1

NAME: VIRAJ TANDEL

TE-IT ROLL : 69

BATCH-C

**TITLE**: Write an encryption and decryption algorithm using Ceaser cipher, playfair cipher and hill cipher

## Theory:

Caesar Cipher:

The Caesar cipher is one of the earliest known and simplest ciphers. It is a type of substitution cipher in which each letter in the plaintext is 'shifted' a certain number of places down the alphabet. For example, with a shift of 1, A would be replaced by B, B would become C, and so on.

Playfair Cipher:

The Playfair cipher was the first practical digraph substitution cipher. The technique encrypts pairs of letters (digraphs), instead of single letters as in the simple substitution cipher. The Playfair is significantly harder to break since the frequency analysis used for simple substitution ciphers does not work with it. Frequency analysis can still be undertaken, but on the 25*25=625 possible digraphs rather than the 25 possible monographs. Frequency analysis thus requires much more ciphertext in order to work.

**Algorithm:**

The 'key' for a playfair cipher is generally a word, for the sake of example we will choose 'monarchy'. This is then used to generate a 'key square', e.g.

```
m o n a r c
h y b d e f
g i k l p q
s t u v w x
```

Any sequence of 25 letters can be used as a key, so long as all letters are in it and there are no repeats. Note that there is no 'j', it is combined with 'i'. We now apply the encryption rules to encrypt the plaintext.

1.Remove any punctuation or characters that are not present in the key square (this may mean spelling out numbers, punctuation etc.).

2.Identify any double letters in the plaintext and replace the second occurence with an 'x' e.g. 'hammer' -> 'hamxer'.

3.If the plaintext has an odd number of characters, append an 'x' to the end to make it even. 4.Break the plaintext into pairs of letters, e.g. 'hamxer' -> 'ha mx er'

5. The algorithm now works on each of the letter pairs.
6. Locate the letters in the key square, (the examples given are using the key square above)

a.If the letters are in different rows and columns, replace the pair with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair. The order is important – the first encrypted letter of the pair is the one that lies on the same row as the first plaintext letter. 'ha' -> 'bo', 'es' -> 'il'

b.If the letters appear on the same row of the table, replace them with the letters to their immediate right respectively (wrapping around to the left side of the row if a letter in the original pair was on the right side of the row). 'ma' -> 'or', 'lp' -> 'pq'
c.If the letters appear on the same column of the table, replace them with the letters immediately below respectively (wrapping around to the top side of the column if a letter in the original pair was on the bottom side of the column). 'rk' -> 'dt', 'pv' -> 'vo'

Hill Cipher:

To encrypt a message using the Hill Cipher we must first turn our keyword into a key matrix (a 2 x 2 matrix for working with digraphs, a 3 x 3 matrix for working with trigraphs, etc). We also turn the plaintext into digraphs (or trigraphs) and each of these into a column vector. We then perform matrix multiplication modulo the length of the alphabet (i.e. 26) on each vector. These vectors are then converted back into letters to produce the ciphertext.

## Code:

Caesar cipher:

```
def encrypt(text,s):
    result=""
    for i in range(len(text)):
            char=text[i]
            result+=chr((ord(char)+ s-97) %26 +97)
    return result
text = "CEASER CIPHER DEMO"
s = 4
print ("Plain Text : " + text)
print ("Shift pattern : " ,s)
print ("Cipher: " +encrypt(text,s).upper())
```

General Ceasor cipher:

```
    def encrypt(text,s):
            result=""
            for i in range(len(text)):
                    char=text[i]
                            #chr->integer-->unicode, ord->reverse of chr
                    result+=chr((ord(char)+ s-97) %26 +97)
            return result

    text = "CEASER CIPHER DEMO"
    s =int(input("Enter the key :"))
    print ("Plain Text : " + text)
    print "Shift pattern : " ,s
    print ("Cipher: " +encrypt(text,s).upper())
```

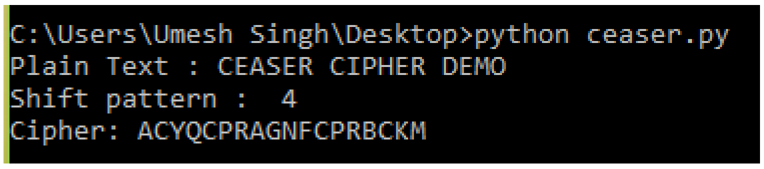
Hill cipher:

```
    message = [[0]*2 for i in range(2)]
```

```
cipher = [[0]*2 for i in range(2)]
key=[[3,3],[2,5]]
def encrypt(message):
        for i in range(2):
                for j in range(2):
                        cipher[i][j] = 0
                for x in range(2):

                        cipher[i][j] += (key[i][x] *  message[x][j])
                        cipher[i][j] = cipher[i][j] % 26
mess = raw_input("Enter Message: ")
k=0
for j in range(2):
        for i in range(2):
                        message[i][j] = ord(mess[k]) % 65
                        k=k+1
                        encrypt(message)
Text = []
for j in range(2):
        for i in range(2):
                Text.append(chr(cipher[i][j] + 65))
print "Cipher: ", "".join(Text)
```

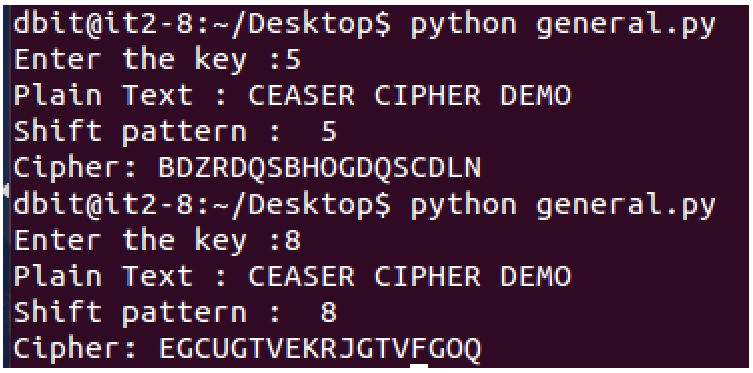
Snapshots:

Caesar cipher:



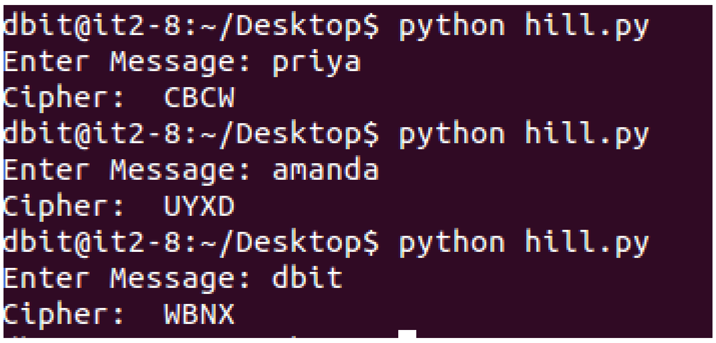
```
C:\Users\Umesh Singh\Desktop>python ceaser.py
Plain Text : CEASER CIPHER DEMO
Shift pattern :  4
Cipher: ACYQCPRAGNFCPRBCKM
```

General Ceasor Cipher:



```
dbit@it2-8:~/Desktop$ python general.py
Enter the key :5
Plain Text : CEASER CIPHER DEMO
Shift pattern :  5
Cipher: BDZRDQSBHOGDQSCDLN
dbit@it2-8:~/Desktop$ python general.py
Enter the key :8
Plain Text : CEASER CIPHER DEMO
Shift pattern :  8
Cipher: EGCUGTVEKRJGTVFGOQ
```

Hill Cipher:



```
dbit@it2-8:~/Desktop$ python hill.py
Enter Message: priya
Cipher:  CBCW
dbit@it2-8:~/Desktop$ python hill.py
Enter Message: amanda
Cipher:  UYXD
dbit@it2-8:~/Desktop$ python hill.py
Enter Message: dbit
Cipher:  WBNX
```

Playfair:

Conclusion: Caesar, Playfair, Hill ciphers were implemented using python.

**DON BOSCO INSTITUTE OF TECHNOLOGY**

DON BOSCO INSTITUTE OF TECHNOLOGY

EXPERIMENT – 2

ITL502.2

NAME: VIRAJ TANDEL

TE-IT ROLL : 69
BATCH-C

TITLE: Implementation of RSA cryptosystem and RSA Digital signature

Theory:
RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes that the Public Key is given to everyone and Private key is kept private.

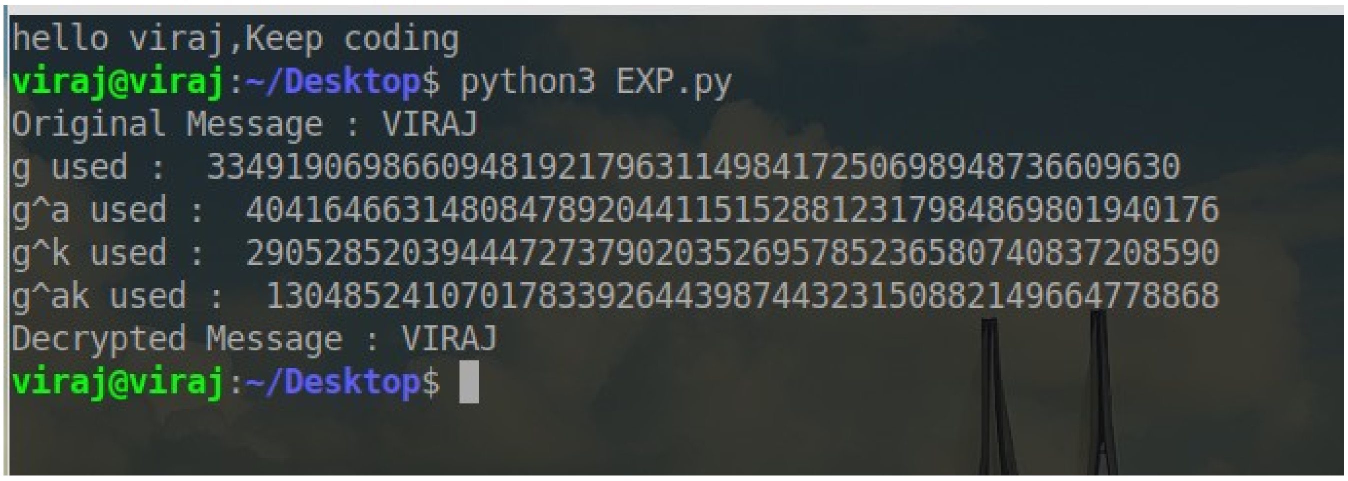
Algorithm:

1. Select two prime numbers p and q.
2. n=p*q
3. Find totient function of n (φ(n))
4. Choose an integer e such that 1 < e < φ(n) and GCD(e, φ(n))=1
5. Determine d such that d * e= 1 (mod φ(n))
6. Here e and n are the Public keys
7. Private keys are d and φ(n)
8. Encryption C=P^e (mod n) , Where P in plaintext and C in ciphertext
9. Decryption P=C^d (mod n) , Where P in plaintext and C in ciphertext

Code:

```
import random
from math import pow

a = random.randint(2, 10)

def gcd(a, b):
    if a < b:
        return gcd(b, a)
    elif a % b == 0:
        return b;
    else:
            return gcd(b, a % b)

    # Generating large random numbers
    def gen_key(q):

        key = random.randint(pow(10, 20), q)
        while gcd(q, key) != 1:
            key = random.randint(pow(10, 20), q)

        return key

    # Modular exponentiation
    def power(a, b, c):
        x = 1
        y = a

        while b > 0:
            if b % 2 == 0:
                x = (x * y) % c;
            y = (y * y) % c
            b = int(b / 2)
        return x % c

# Asymmetric encryption
def encrypt(msg, q, h, g):

    en_msg = []

    k = gen_key(q)# Private key for sender
    s = power(h, k, q)
    p = power(g, k, q)

    for i in range(0, len(msg)):
        en_msg.append(msg[i])

    print("g^k used : ", p)
    print("g^ak used : ", s)
    for i in range(0, len(en_msg)):
        en_msg[i] = s * ord(en_msg[i])

    return en_msg, p

def decrypt(en_msg, p, key, q):

    dr_msg = []
    h = power(p, key, q)
    for i in range(0, len(en_msg)):
        dr_msg.append(chr(int(en_msg[i]/h)))

    return dr_msg

# Driver code
def main():
    msg = 'VIRAJ'
    print("Original Message :", msg)

    q = random.randint(pow(10, 20), pow(10, 50))
    g = random.randint(2, q)

    key = gen_key(q)# Private key for receiver
    h = power(g, key, q)
    print("g used : ", g)
    print("g^a used : ", h)

    en_msg, p = encrypt(msg, q, h, g)
    dr_msg = decrypt(en_msg, p, key, q)
    dmsg = ''.join(dr_msg)
    print("Decrypted Message :", dmsg);


if __name__ == '__main__':
    main()
```

Snapshots:



```
hello viraj,Keep coding
viraj@viraj:~/Desktop$ python3 EXP.py
Original Message : VIRAJ
g used :  33491906986609481921796311498417250698948736609630
g^a used :  40416466314808478920441151528812317984869801940176
g^k used :  29052852039444727379020352695785236580740837208590
g^ak used :  13048524107017833926443987443231508821496647788 68
Decrypted Message : VIRAJ
viraj@viraj:~/Desktop$
```

Conclusion:

RSA algorithm was implemented using python.

DON BOSCO INSTITUTE OF TECHNOLOGY

EXPERIMENT – 3

ITL502.2

NAME: VIRAJ TANDEL

TE-IT ROLL : 69
BATCH-C

TITLE: Implementation of Diffie Hellman Key exchange algorithm

Theory:

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

Algorithm:

1. The two parties need to choose two numbers p and g (these no. are public).

   • p is large prime number. • g is random number.

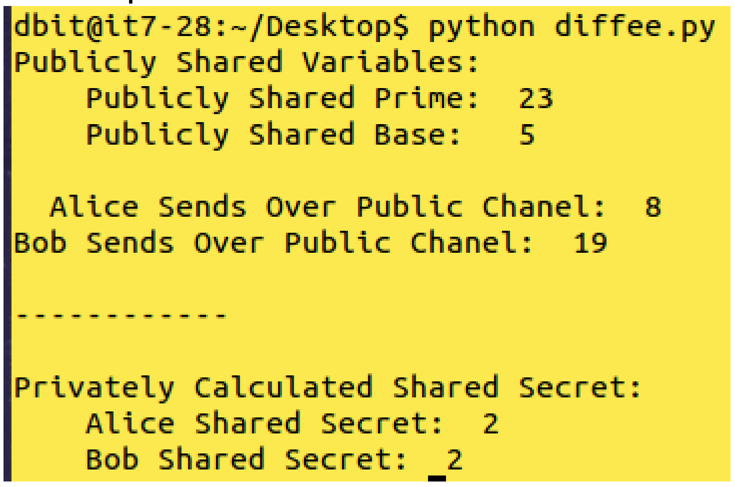
2. Sender chooses a large random number “x” and calculates. R1=g^x mod p
3. Receiver chooses a another large random number “y” and calculates R2 = g^y mod p
4. Sender sends R1 to Receiver.
5. Receiver sends R2 to Sender
6. Sender calculates K = (R2)^x mod p
7. Receiver also calculates K = (R1)^y mod p
8. Both Sender and Receiver have K which is a symmetric key

Code:

```
from __future__ import print_function
UsedsharedPrime = 23
print( "Publicly Shared Variables:")
print( "    Publicly Shared Prime: ",sharedPrime )
print( "    Publicly Shared Base:  " , sharedBase )
```

```
# Alice Sends Bob
A = g^a mod p
A = (sharedBase**aliceSecret) % sharedPrime
print( "\n  Alice Sends Over Public Chanel: " , A )
# Bob Sends Alice
B = g^b mod p
B = (sharedBase ** bobSecret) % sharedPrime
print("Bob Sends Over Public Chanel: ", B )
print( "\n------------\n" )
print("Privately Calculated Shared Secret:"
# Alice Computes Shared Secret:
s = B^a mod p
aliceSharedSecret = (B ** aliceSecret) % sharedPrime
print("    Alice Shared Secret: ", aliceSharedSecret )
# Bob Computes Shared Secret:
s = A^b mod p
bobSharedSecret = (A**bobSecret) % sharedPrime
print( "    Bob Shared Secret: ", bobSharedSecret )
```

Snapshot:



```
dbit@it7-28:~/Desktop$ python diffee.py
Publicly Shared Variables:
    Publicly Shared Prime:  23
    Publicly Shared Base:   5

  Alice Sends Over Public Chanel:  8
Bob Sends Over Public Chanel:  19

------------

Privately Calculated Shared Secret:
    Alice Shared Secret:  2
    Bob Shared Secret:  2
```

Conclusion:

Deffie Hellman was understood and implemented using Python. Both the cases where p was a prime number and not a prime number was checked.

DON BOSCO INSTITUTE OF TECHNOLOGY

## <u>EXPERIMENT – 4</u>

<u>ITL502.3</u>

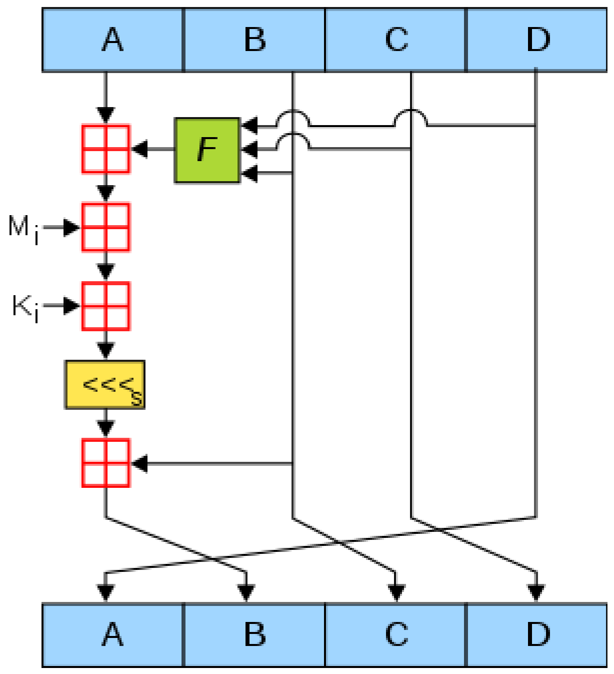
NAME: PRIYA SINGH

TE-IT ROLL : 67
BATCH-C

TITLE: Write program to demonstrate integrity management by implementing message digest using MD5

Theory:
MD5 (Message Digest algorithm 5) is a widely used cryptographic hash function with a 128bit hash value. An MD5 hash is typically expressed as a 32 digit hexadecimal number. MD5 processes a variable length message into a fixed length output of 128 bits. The input message is broken up into chunks of 512 bit blocks (sixteen 32bit little endian integers) ; The message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits less than a multiple of 512. The remaining bits are filled up with a 64bit integer representing the length of the original message, in bits.

The main MD5 algorithm operates on a 128bit state, divided into four 32bit words, denoted A, B, C and D. These are initialized to certain fixed constants. The main algorithm then operates on each 512bit message block in turn, each block modifying the state. The processing of a message block consists of four similar stages, termed rounds; each round is composed of 16 similar operations based on a nonlinear function F, modular addition, and left rotation.



Algorithm:

1. Append Padding Bits :The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo
512. Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

2. Append Length :

A 64bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^64, then only the loworder 64 bits of b are used. (These bits are appended as two 32bit words and appended loworder word first in accordance with the previous conventions.) At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32 bit) words. Let M[0 ... N1] denote the words of the resulting message, where N is a multiple of 16.

3. Initialize MD Buffer :

A fourword buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32bit register. These registers are initialized to the following values in hexadecimal, loworder bytes first):

4. Process Message in 16Word Blocks :

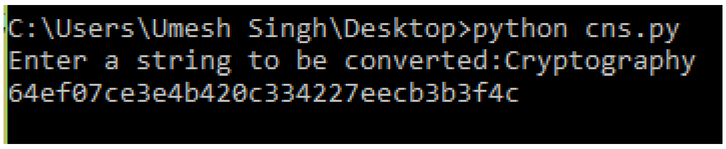
We first define four auxiliary functions that each take as input three 32bit words and produce as output one 32bit word.

5. Output :

The message digest produced as output is A, B, C, D. That is, we begin with the low order byte of A, and end with the highorder byte of D.
Code:

```
import hashlib

string = input("enter a string to be converted")

result=(hashlib.md5(string.encode()))

print(result.hexdigest())
```

Snapshots:



```
C:\Users\Umesh Singh\Desktop>python cns.py
Enter a string to be converted:Cryptography
64ef07ce3e4b420c334227eecb3b3f4c
```

Conclusion: MD5 was implemented in python using hashlib module.

DON BOSCO INSTITUTE OF TECHNOLOGY

EXPERIMENT – 5

ITL502.5

NAME: VIRAJ TANDEL

TE-IT ROLL : 69
BATCH-C

TITLE: Use of different options in NMAP to scan open ports, perform OS fingerprinting, do a ping scan, tcp port scan, udp port scan.

Theory:
Nmap (Network Mapper) is a security scanner originally written by Gordon Lyon (also known by his pseudonym Fyodor Vaskovich) used to discover hosts and services on a computer network, thus creating a "map" of the network. To accomplish its goal, Nmap sends specially crafted packets to the target host and then analyzes the responses. Unlike many simple port scanners that just send packets at some predefined constant rate, Nmap accounts for the network conditions (latency fluctuations, network congestion, the target interference with the scan) during the run. Also, owing to the large and active user community providing feedback and contributing to its features, Nmap has been able to extend its discovery capabilities beyond simply figuring out whether a host is up or down and which ports are open and closed; it can determine the operating system of the target, names and versions of the listening services, estimated uptime, type of device, and presence of a firewall.

Nmap features include:

1. Host Discovery – Identifying hosts on a network. For example, listing the hosts which respond to pings or have a particular port open.
2. Port Scanning – Enumerating the open ports on one or more target hosts.
3. Version Detection – Interrogating listening network services listening on remote devices to determine the application name and version number.
4. OS Detection – Remotely determining the operating system and some hardware characteristics of network devices.

Basic commands working in Nmap:

1. For target specifications: nmap <URL or IP address>
2. For OS detection: nmap -O <target's URL or IP>
3. For version detection: nmap -sV <target's URL or IP>

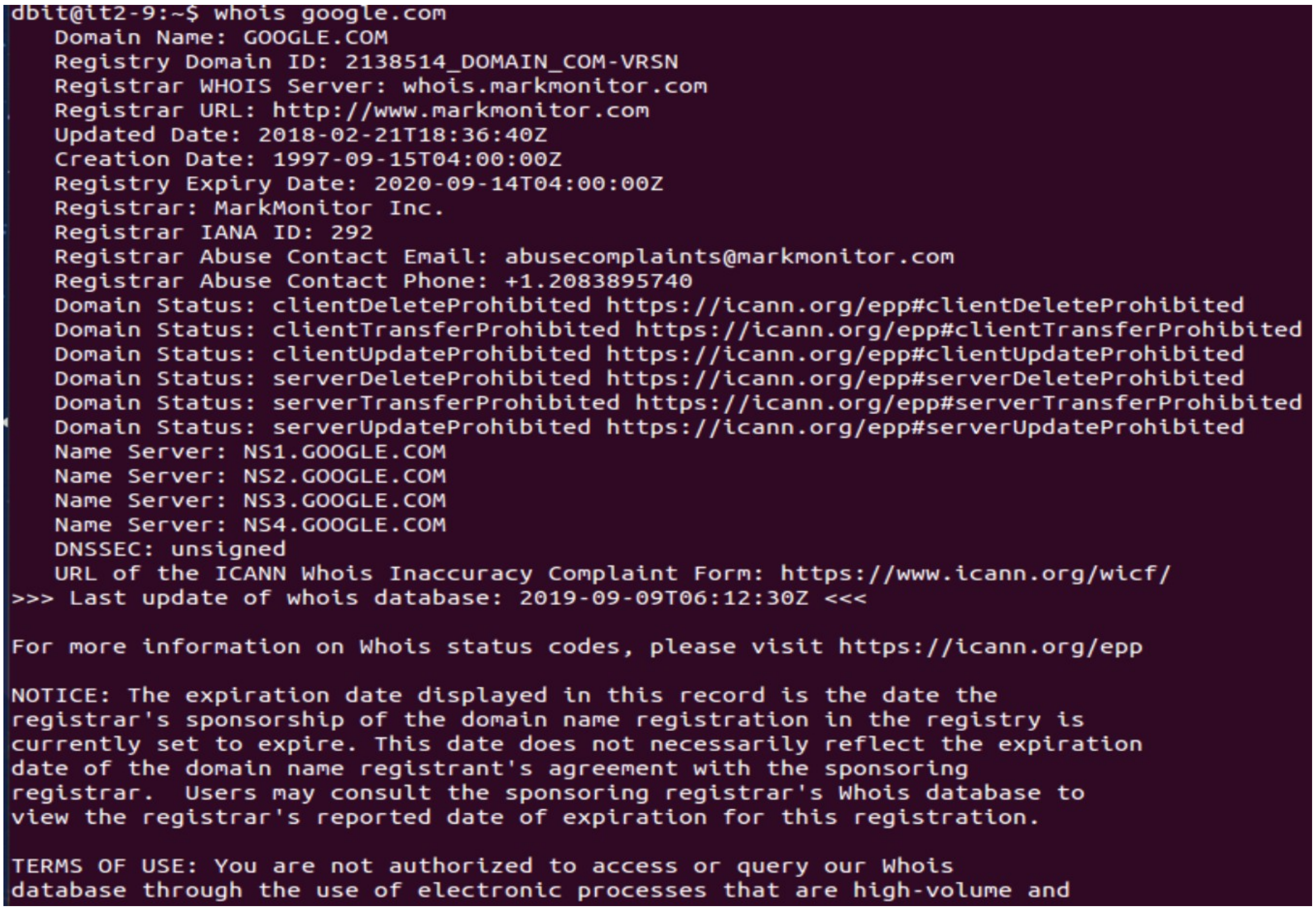
SYN scan is the default and most popular scan option for good reasons. It can be performed quickly, scanning thousands of ports per second on a fast network not hampered by restrictive firewalls. It is also relatively unobtrusive and stealthy since it never completes TCP connections.

Procedure:

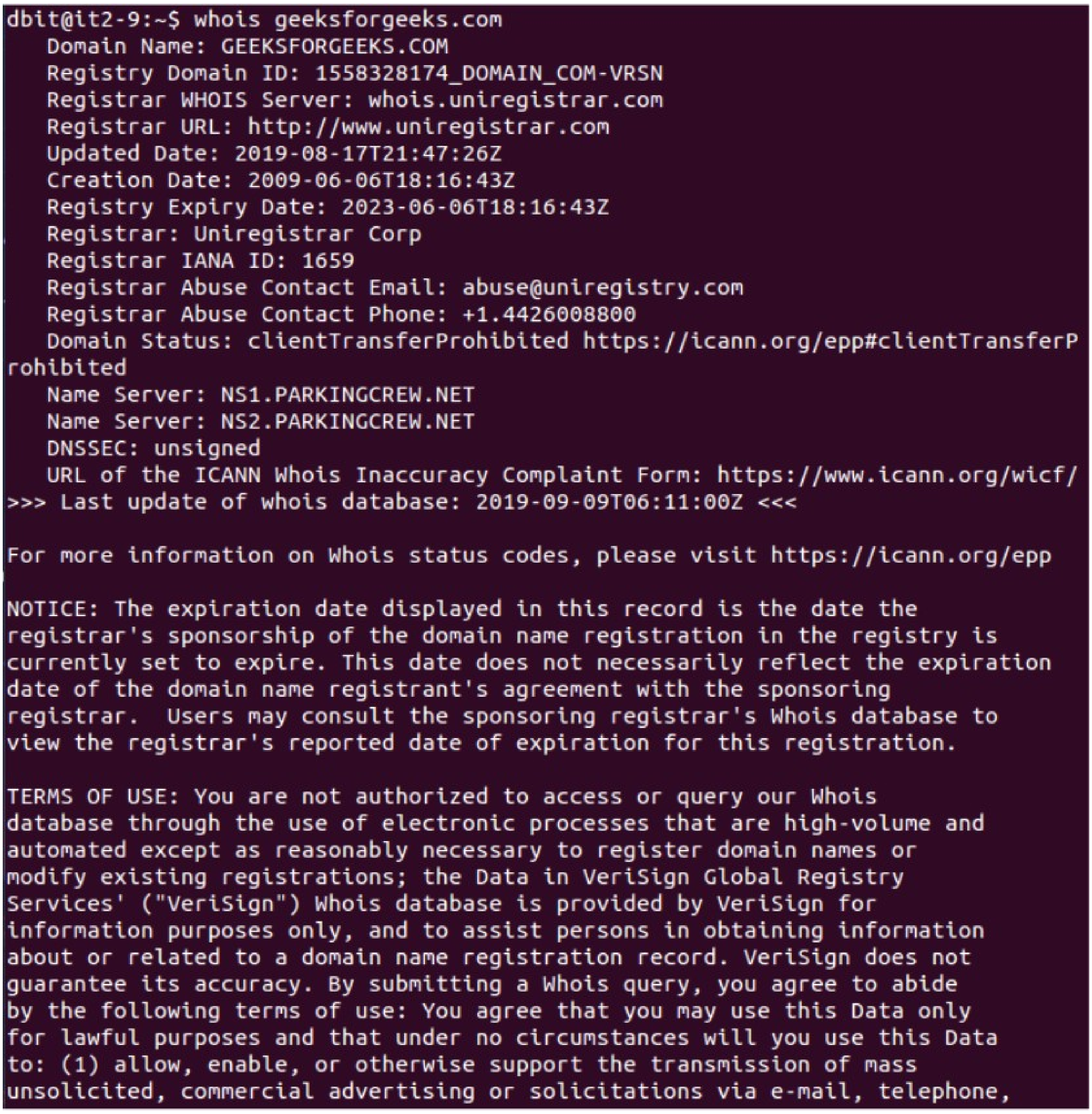
1. Installation – sudo apt install nmap
2. Obtain IP address using ifconfig
3. Perforing a scan of the local Network
   a) nmap -sP 192.168.1.*

Snapshots:

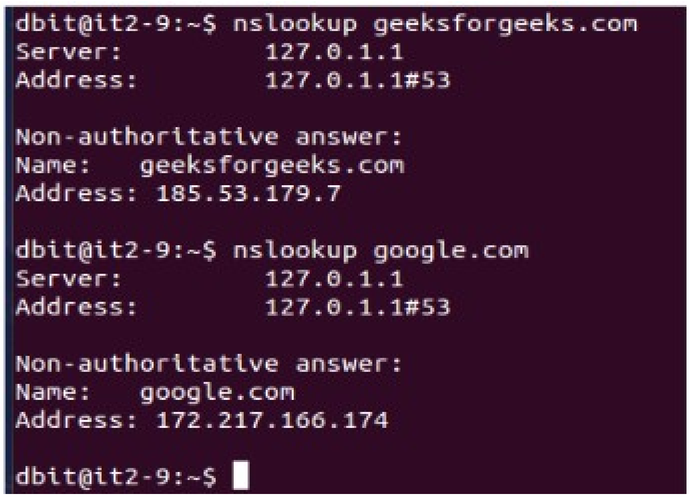
WHOIS



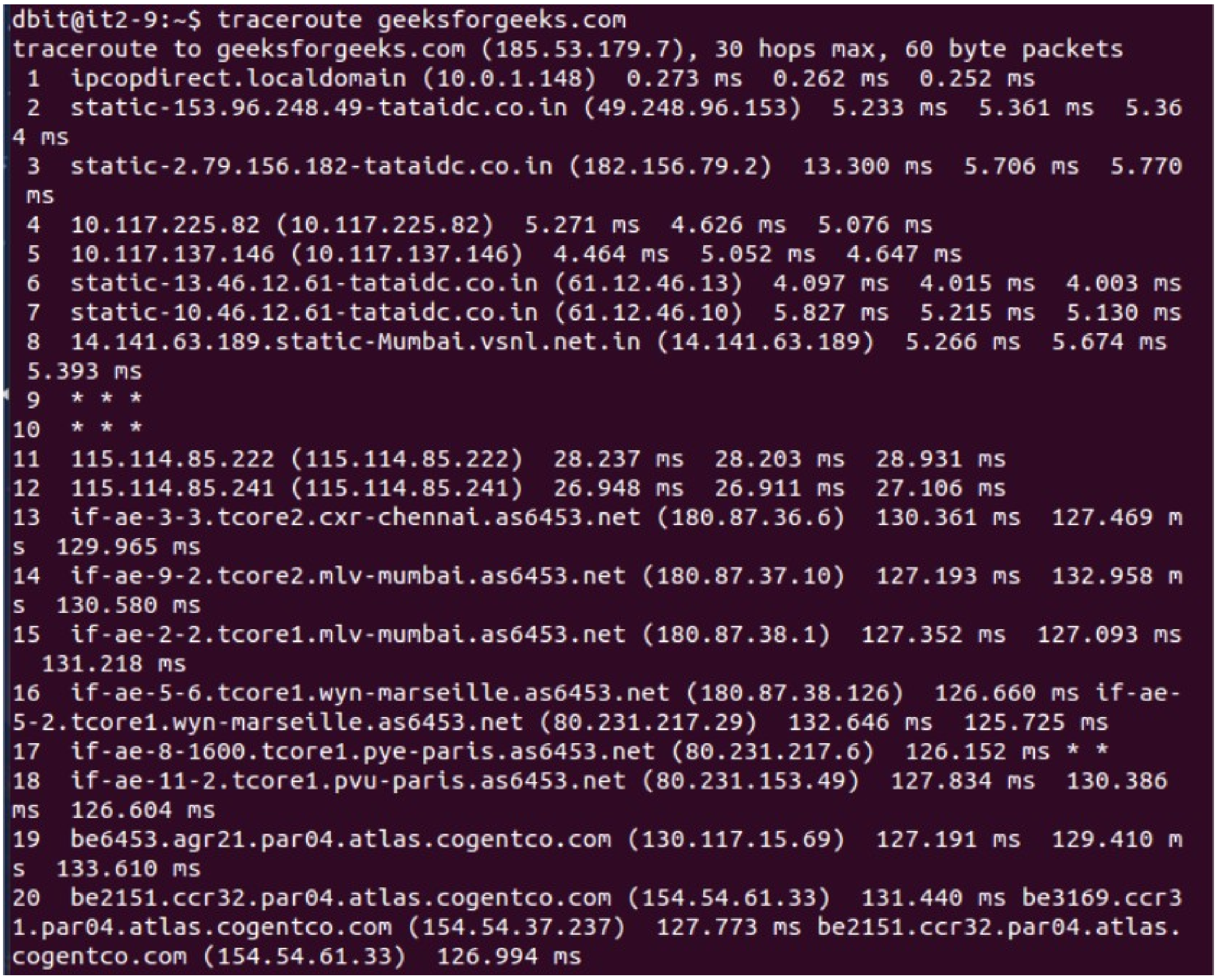
```
dbit@it2-9:~$ whois google.com
   Domain Name: GOOGLE.COM
   Registry Domain ID: 2138514_DOMAIN_COM-VRSN
   Registrar WHOIS Server: whois.markmonitor.com
   Registrar URL: http://www.markmonitor.com
   Updated Date: 2018-02-21T18:36:40Z
   Creation Date: 1997-09-15T04:00:00Z
   Registry Expiry Date: 2020-09-14T04:00:00Z
   Registrar: MarkMonitor Inc.
   Registrar IANA ID: 292
   Registrar Abuse Contact Email: abusecomplaints@markmonitor.com
   Registrar Abuse Contact Phone: +1.2083895740
   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: 2019-09-09T06:12:30Z <<<

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
```



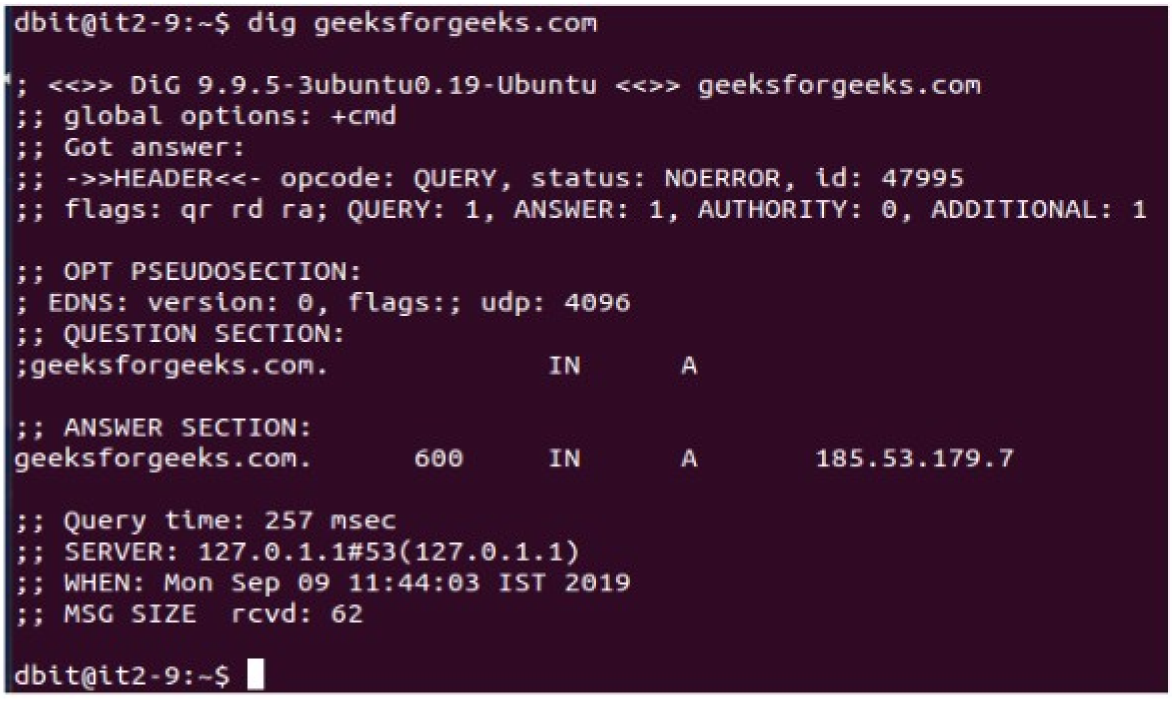
```
dbit@it2-9:~$ whois geeksforgeeks.com
   Domain Name: GEEKSFORGEEKS.COM
   Registry Domain ID: 1558328174_DOMAIN_COM-VRSN
   Registrar WHOIS Server: whois.uniregistrar.com
   Registrar URL: http://www.uniregistrar.com
   Updated Date: 2019-08-17T21:47:26Z
   Creation Date: 2009-06-06T18:16:43Z
   Registry Expiry Date: 2023-06-06T18:16:43Z
   Registrar: Uniregistrar Corp
   Registrar IANA ID: 1659
   Registrar Abuse Contact Email: abuse@uniregistry.com
   Registrar Abuse Contact Phone: +1.4426008800
   Domain Status: clientTransferProhibited https://icann.org/epp#clientTransferP
rohibited
   Name Server: NS1.PARKINGCREW.NET
   Name Server: NS2.PARKINGCREW.NET
   DNSSEC: unsigned
   URL of the ICANN Whois Inaccuracy Complaint Form: https://www.icann.org/wicf/
>>> Last update of whois database: 2019-09-09T06:11:00Z <<<

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,
```



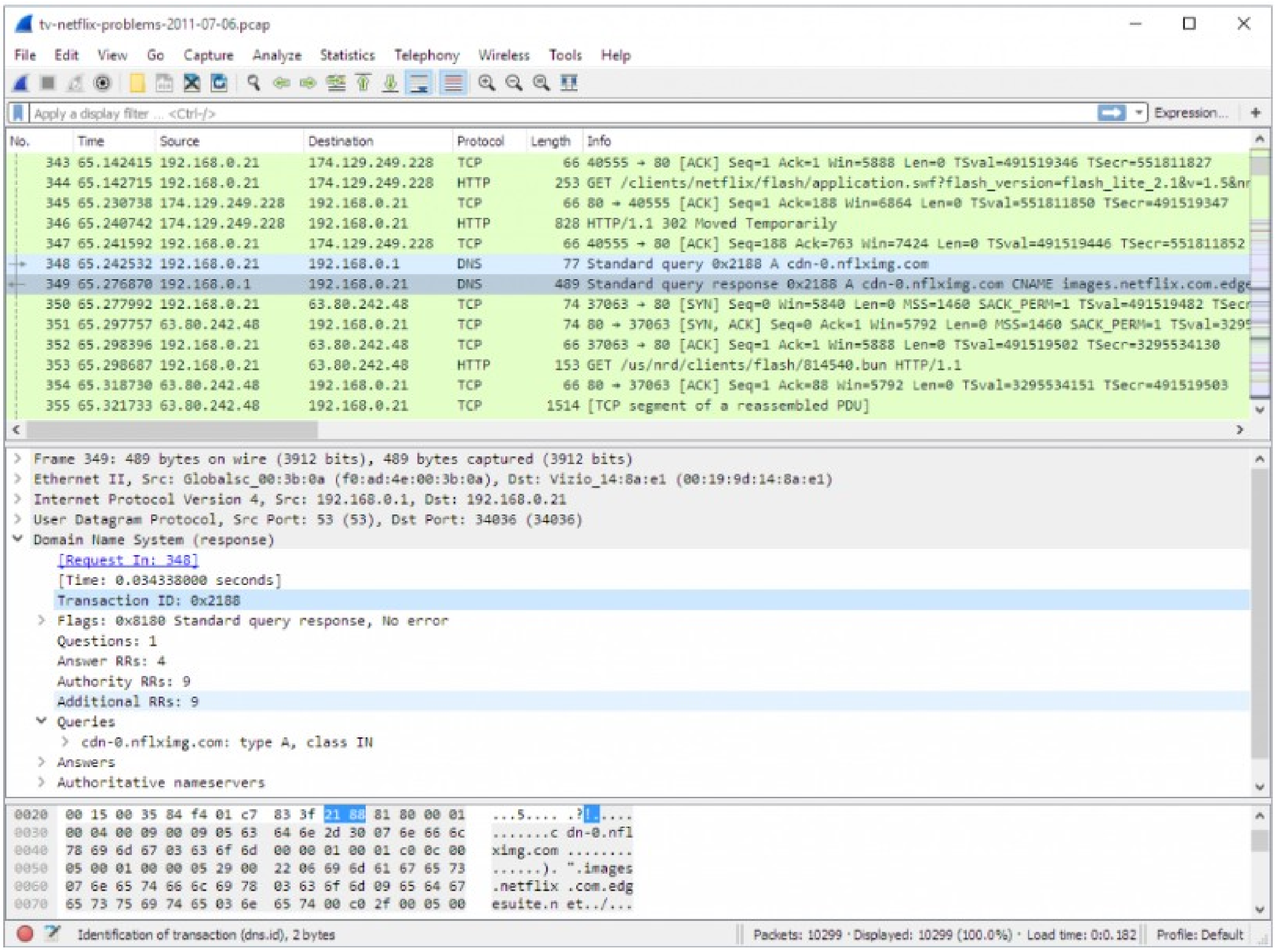
```
dbit@it2-9:~$ nslookup geeksforgeeks.com
Server:         127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
Name:   geeksforgeeks.com
Address: 185.53.179.7

dbit@it2-9:~$ nslookup google.com
Server:         127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
Name:   google.com
Address: 172.217.166.174

dbit@it2-9:~$ 
```



```
dbit@it2-9:~$ traceroute geeksforgeeks.com
traceroute to geeksforgeeks.com (185.53.179.7), 30 hops max, 60 byte packets
 1  ipcopdirect.localdomain (10.0.1.148)  0.273 ms  0.262 ms  0.252 ms
 2  static-153.96.248.49-tataidc.co.in (49.248.96.153)  5.233 ms  5.361 ms  5.36
4 ms
 3  static-2.79.156.182-tataidc.co.in (182.156.79.2)  13.300 ms  5.706 ms  5.770
 ms
 4  10.117.225.82 (10.117.225.82)  5.271 ms  4.626 ms  5.076 ms
 5  10.117.137.146 (10.117.137.146)  4.464 ms  5.052 ms  4.647 ms
 6  static-13.46.12.61-tataidc.co.in (61.12.46.13)  4.097 ms  4.015 ms  4.003 ms
 7  static-10.46.12.61-tataidc.co.in (61.12.46.10)  5.827 ms  5.215 ms  5.130 ms
 8  14.141.63.189.static-Mumbai.vsnl.net.in (14.141.63.189)  5.266 ms  5.674 ms
 5.393 ms
 9  * * *
10  * * *
11  115.114.85.222 (115.114.85.222)  28.237 ms  28.203 ms  28.931 ms
12  115.114.85.241 (115.114.85.241)  26.948 ms  26.911 ms  27.106 ms
13  if-ae-3-3.tcore2.cxr-chennai.as6453.net (180.87.36.6)  130.361 ms  127.469 m
s  129.965 ms
14  if-ae-9-2.tcore2.mlv-mumbai.as6453.net (180.87.37.10)  127.193 ms  132.958 m
s  130.580 ms
15  if-ae-2-2.tcore1.mlv-mumbai.as6453.net (180.87.38.1)  127.352 ms  127.093 ms
  131.218 ms
16  if-ae-5-6.tcore1.wyn-marseille.as6453.net (180.87.38.126)  126.660 ms if-ae-
5-2.tcore1.wyn-marseille.as6453.net (80.231.217.29)  132.646 ms  125.725 ms
17  if-ae-8-1600.tcore1.pye-paris.as6453.net (80.231.217.6)  126.152 ms * *
18  if-ae-11-2.tcore1.pvu-paris.as6453.net (80.231.153.49)  127.834 ms  130.386
ms  126.604 ms
19  be6453.agr21.par04.atlas.cogentco.com (130.117.15.69)  127.191 ms  129.410 m
s  133.610 ms
20  be2151.ccr32.par04.atlas.cogentco.com (154.54.61.33)  131.440 ms be3169.ccr3
1.par04.atlas.cogentco.com (154.54.37.237)  127.773 ms be2151.ccr32.par04.atlas.
cogentco.com (154.54.61.33)  126.994 ms
```



```
dbit@it2-9:~$ dig geeksforgeeks.com

; <<>> DiG 9.9.5-3ubuntu0.19-Ubuntu <<>> geeksforgeeks.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 47995
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;geeksforgeeks.com.             IN      A

;; ANSWER SECTION:
geeksforgeeks.com.      600     IN      A       185.53.179.7

;; Query time: 257 msec
;; SERVER: 127.0.1.1#53(127.0.1.1)
;; WHEN: Mon Sep 09 11:44:03 IST 2019
;; MSG SIZE  rcvd: 62

dbit@it2-9:~$
```


tv-netflix-problems-2011-07-06.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.   Time       Source          Destination      Protocol Length Info
  343 65.142415 192.168.0.21    174.129.249.228  TCP        66 40555 → 80 [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSval=491519346 TSecr=551811827
  344 65.142715 192.168.0.21    174.129.249.228  HTTP      253 GET /clients/netflix/flash/application.swf?flash_version=flash_lite_2.1&v=1.5&nr
  345 65.230738 174.129.249.228 192.168.0.21     TCP        66 80 → 40555 [ACK] Seq=1 Ack=188 Win=6864 Len=0 TSval=551811850 TSecr=491519347
  346 65.240742 174.129.249.228 192.168.0.21     HTTP      828 HTTP/1.1 302 Moved Temporarily
  347 65.241592 192.168.0.21    174.129.249.228  TCP        66 40555 → 80 [ACK] Seq=188 Ack=763 Win=7424 Len=0 TSval=491519446 TSecr=551811852
  348 65.242532 192.168.0.21    192.168.0.1      DNS        77 Standard query 0x2188 A cdn-0.nflximg.com
  349 65.276870 192.168.0.1     192.168.0.21     DNS       489 Standard query response 0x2188 A cdn-0.nflximg.com CNAME images.netflix.com.edge
  350 65.277992 192.168.0.21    63.80.242.48     TCP        74 37063 → 80 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=491519482 TSecr
  351 65.297757 63.80.242.48    192.168.0.21     TCP        74 80 → 37063 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=3295
  352 65.298396 192.168.0.21    63.80.242.48     TCP        66 37063 → 80 [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSval=491519502 TSecr=3295534130
  353 65.298687 192.168.0.21    63.80.242.48     HTTP      153 GET /us/nrd/clients/flash/814540.bun HTTP/1.1
  354 65.318730 63.80.242.48    192.168.0.21     TCP        66 80 → 37063 [ACK] Seq=1 Ack=88 Win=5792 Len=0 TSval=3295534151 TSecr=491519503
  355 65.321733 63.80.242.48    192.168.0.21     TCP      1514 [TCP segment of a reassembled PDU]
```

```
> Frame 349: 489 bytes on wire (3912 bits), 489 bytes captured (3912 bits)
> Ethernet II, Src: Globalsc_00:3b:0a (f0:ad:4e:00:3b:0a), Dst: Vizio_14:8a:e1 (00:19:9d:14:8a:e1)
> Internet Protocol Version 4, Src: 192.168.0.1, Dst: 192.168.0.21
> User Datagram Protocol, Src Port: 53 (53), Dst Port: 34036 (34036)
v Domain Name System (response)
    [Request In: 348]
    [Time: 0.034338000 seconds]
    Transaction ID: 0x2188
  > Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 4
    Authority RRs: 9
    Additional RRs: 9
  v Queries
    > cdn-0.nflximg.com: type A, class IN
  > Answers
  > Authoritative nameservers
```

```
0020  00 15 00 35 84 f4 01 c7  83 3f 21 88 81 80 00 01   ...5.... .?!.....
0030  00 04 00 09 00 09 05 63  64 6e 2d 30 07 6e 66 6c   .......c dn-0.nfl
0040  78 69 6d 67 03 63 6f 6d  00 00 01 00 01 c0 0c 00   ximg.com ........
0050  05 00 01 00 00 05 29 00  22 06 69 6d 61 67 65 73   ......). ".images
0060  07 6e 65 74 66 6c 69 78  03 63 6f 6d 09 65 64 67   .netflix .com.edg
0070  65 73 75 69 74 65 03 6e  65 74 00 c0 2f 00 05 00   esuite.n et../...
```

```
Identification of transaction (dns.id), 2 bytes    Packets: 10299 · Displayed: 10299 (100.0%) · Load time: 0:0.182    Profile: Default
```

Conclusion:We have studied the use of network reconnaissance tools like WHOIS, dig, traceroute, nslookup to gather information about networks and domain registrars.

References: Cryptography & Network Security, William Stallings, Pearson, 6 th Edition.