Course Code : CSL604 **Course Name : System Security Lab**

Class : TE-CO **Batch : Computer Engineering**

Roll no : 18CO63 **Name : SHAIKH TAUSEEF MUSHTAQUE ALI**

Experiment : 01

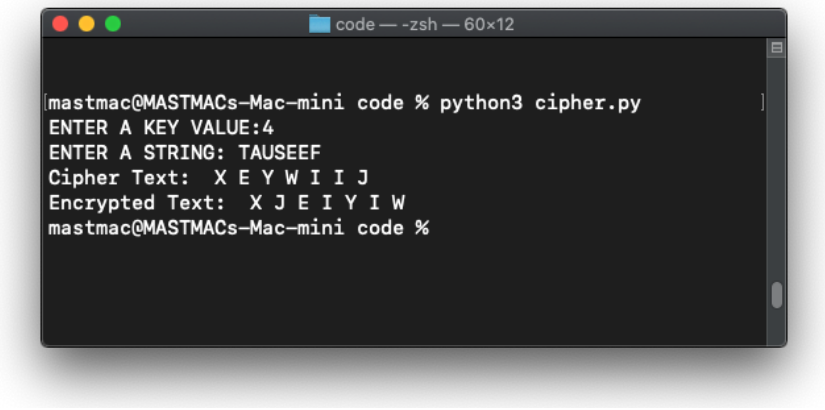
Aim : Design and Implementation of a product cipher using Substitution and Transposition ciphers

Code :

```
import string

k=int(input("ENTER A KEY VALUE:"))
d=str(input("ENTER A STRING: "))
ct = []
alphabets = string.ascii_uppercase
for j in d:
        b=j.upper()
        if b in alphabets and j.islower():
                e=(alphabets.index(b)+k)%26
                ct.append(alphabets[e].lower())
        elif b in alphabets and j.isupper():
                a=(alphabets.index(b)+k)%26
                ct.append(alphabets[a].upper())
        else:
                ct.append(" ")
matrix = [[False for i in range(len(ct))]
 for j in range(k)]

print("Cipher Text: ",*ct)
j=0
for i in range(len(ct)):
   matrix[j][i]=ct[i]
   if j == k - 1:
      flag = False
   elif j == 0:
      flag = True
   if flag == True:
      j = j + 1
   else:
      j = j - 1
answer=[]
for key in range(k):
   for text in range(len(ct)):
      if matrix[key][text]!=False:
         answer.append(matrix[key][text])
```

```
print("Encrypted Text: ", *answer)
```

Output :



Conclusion:

A **product cipher** combines two or more transformations in a manner intending that the resulting cipher
is more secure than the individual components to make it resistant to cryptanalysis. Implemented
product cipher using Substitution and Transposition ciphers

Course Code : CSL604 **Course Name : System Security Lab**

Class : TE-CO **Batch : Computer Engineering**

Roll no : 18CO63 **Name : SHAIKH TAUSEEF MUSHTAQUE ALI**

Experiment : 02

Aim : Implementation and analysis of Playfair cipher.

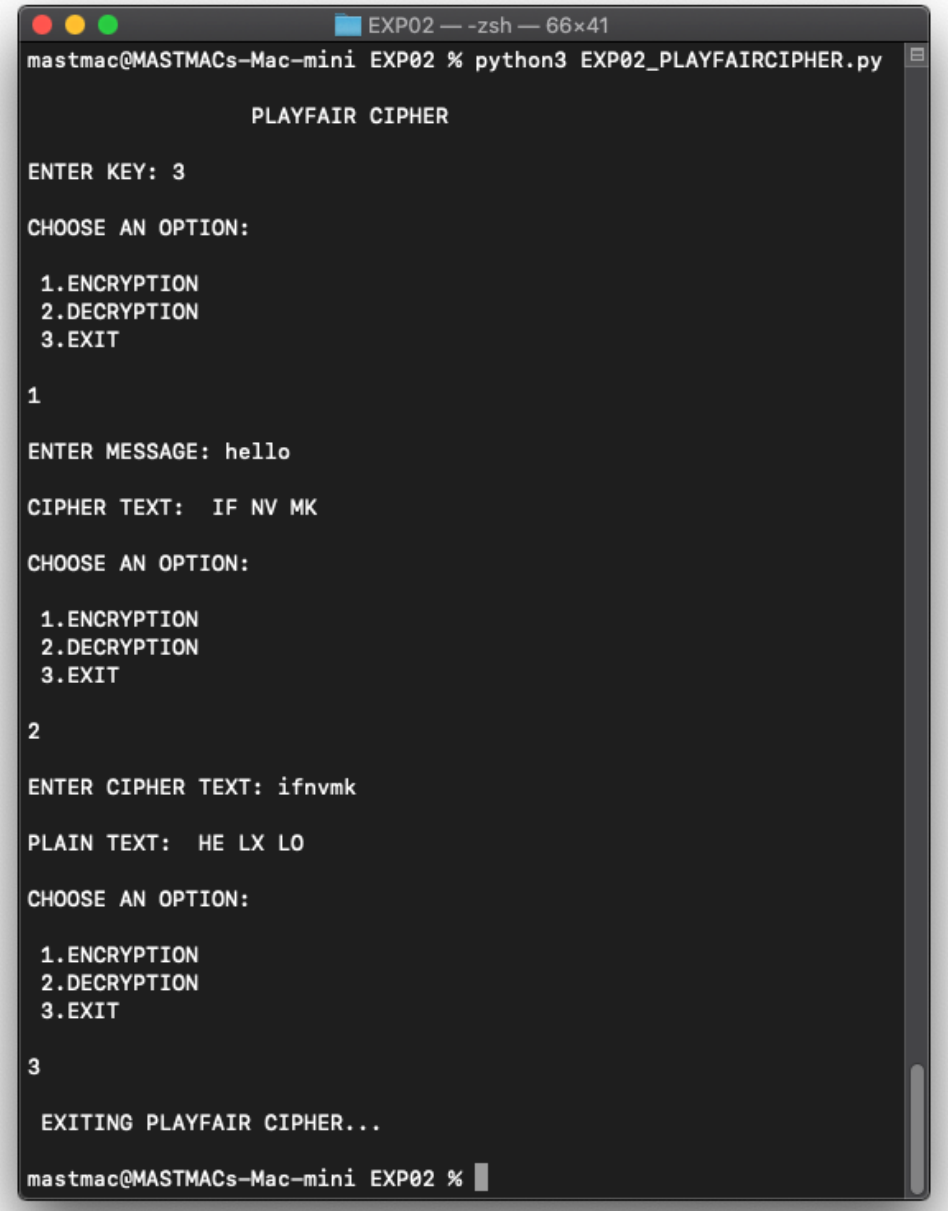
Code :

```
print("\n\t\t PLAYFAIR CIPHER \n")

key=input("ENTER KEY: ")
key=key.replace(" ", "")
key=key.upper()
def matrix(x,y,initial):
    return [[initial for i in range(x)] for j in range(y)]

result=list()
for c in key:
    if c not in result:
        if c=='J':
            result.append('I')
        else:
            result.append(c)
flag=0
for i in range(65,91):
    if chr(i) not in result:
        if i==73 and chr(74) not in result:
            result.append("I")
            flag=1
        elif flag==0 and i==73 or i==74:
            pass
        else:
            result.append(chr(i))
k=0
my_matrix=matrix(5,5,0)
for i in range(0,5):
    for j in range(0,5):
        my_matrix[i][j]=result[k]
        k+=1

def locindex(c):
    loc=list()
    if c=='J':
        c='I'
```

```
    for i ,j in enumerate(my_matrix):
        for k,l in enumerate(j):
            if c==l:
                loc.append(i)
                loc.append(k)
                return loc

def encrypt():
    msg=str(input("\nENTER MESSAGE: "))
    msg=msg.upper()
    msg=msg.replace(" ", "")
    i=0
    for s in range(0,len(msg)+1,2):
        if s<len(msg)-1:
            if msg[s]==msg[s+1]:
                msg=msg[:s+1]+'X'+msg[s+1:]
    if len(msg)%2!=0:
        msg=msg[:]+'X'
    print("\nCIPHER TEXT: ",end=' ')
    while i<len(msg):
        loc=list()
        loc=locindex(msg[i])
        loc1=list()
        loc1=locindex(msg[i+1])
        if loc[1]==loc1[1]:

print("{}{}".format(my_matrix[(loc[0]+1)%5][loc[1]],my_matrix[(loc1[0]+1)%5][loc1[1]]),end=' ')
        elif loc[0]==loc1[0]:

print("{}{}".format(my_matrix[loc[0]][(loc[1]+1)%5],my_matrix[loc1[0]][(loc1[1]+1)%5]),end=' ')
        else:
            print("{}{}".format(my_matrix[loc[0]][loc1[1]],my_matrix[loc1[0]][loc[1]]),end=' ')
        i=i+2

    print("")

def decrypt():
    msg=str(input("\nENTER CIPHER TEXT: "))
    msg=msg.upper()
    msg=msg.replace(" ", "")
    print("\nPLAIN TEXT: ",end=' ')
    i=0
    while i<len(msg):
        loc=list()
        loc=locindex(msg[i])
        loc1=list()
```

```
        loc1=locindex(msg[i+1])
        if loc[1]==loc1[1]:
            print("{}{}".format(my_matrix[(loc[0]-1)%5][loc[1]],my_matrix[(loc1[0]-
1)%5][loc1[1]]),end=' ')
        elif loc[0]==loc1[0]:
            print("{}{}".format(my_matrix[loc[0]][(loc[1]-1)%5],my_matrix[loc1[0]][(loc1[1]-
1)%5]),end=' ')
        else:
            print("{}{}".format(my_matrix[loc[0]][loc1[1]],my_matrix[loc1[0]][loc[1]]),end=' ')
        i=i+2

    print("")

while(1):
    print("\nCHOOSE AN OPTION: \n")
    choice=int(input(" 1.ENCRYPTION \n 2.DECRYPTION \n 3.EXIT \n\n"))
    if choice==1:
        encrypt()
    elif choice==2:
        decrypt()
    elif choice==3:
        print("\n EXITING PLAYFAIR CIPHER... \n")
        exit()
    else:
        print("\nINVALID OPTION! CHOOSE CORRECT OPTION \n")
```

Output:



Conclusion:

A playfair cipher unlike traditional cipher we encrypt a pair of alphabets(digraphs) instead of a single alphabet.

Course Code : CSL604 **Course Name : System Security Lab**

Class : TE-CO **Batch : Computer Engineering**

Roll no : 18CO63 **Name : SHAIKH TAUSEEF MUSHTAQUE ALI**

Experiment : 03

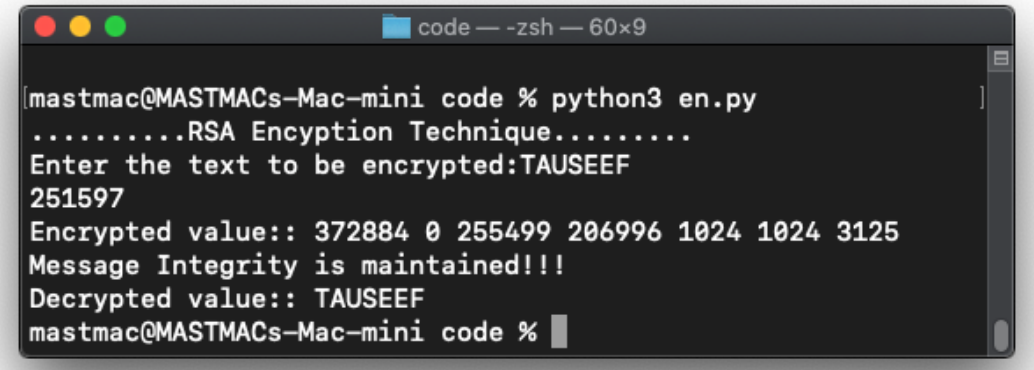
Aim : Implementation and analysis of RSA cryptosystem and Digital signature scheme using RSA.

Code :

```
import hashlib
import random
import math
print(".....RSA Encyption Technique.....")
pt=input("Enter the text to be encrypted:")
code = hashlib.sha1(pt.encode())
code = code.hexdigest()
plain = pt.replace(" ","")
if plain.isalpha():
   pta=pt.lower()
   ptn=[ord(i)%97 for i in pta]
elif pt.isdigit():
   ptn=int(pt)

#n=int(input("Enter a composite prime number(n)"))
primes = []
for x in range(1,1001):
        for y in range(2,x):
                if x%y==0:
                        break
        else:
                primes.append(x)

p, q = random.choice(primes), random.choice(primes)
phi=(p-1)*(q-1)
n = p * q
e= 0
for i in range(2,26):
   if math.gcd(i,phi)==1:
      e=i
      break
def modInverse(a,m):
   for x in range(1, m):
      if (((a%m) * (x%m)) % m == 1):
         return x
   return -1
d = modInverse(e,phi)
print(d)
if d!= -1:
   if plain.isalpha():
```

```
        ct= [(i**e)%n for i in ptn]
        print("Encrypted value::",*ct)
        dt= [(i**d)%n for i in ct]
    else:
        ct = (ptn**e) % n
        dt = (ct**d) % n
else:
    print("Encryption is not Possible!!!!!!!!")

if plain.isalpha() and pt.islower():
    dt = "".join([chr(int(i)+97) for i in dt]).replace("\x81"," ")
elif plain.isalpha() and pt.isupper():
    dt = "".join([chr(65+int(i)) for i in dt]).replace("a"," ")
else:
    dt= str(dt)

hashvalue = hashlib.sha1(dt.encode())
hashvalue = hashvalue.hexdigest()
if code==hashvalue:
    print("Message Integrity is maintained!!!")
    print("Decrypted value::",dt)
else:
    print("Corrupted message!!")
```

Output:



Conclusion:

Implemented and analyzed RSA cryptosystem and Digital signature scheme using RSA.

Course Code : CSL604 **Course Name : System Security Lab**

Class : TE-CO **Batch : Computer Engineering**

Roll no : 18CO63 **Name : SHAIKH TAUSEEF MUSHTAQUE ALI**

Experiment : 04

Aim : For varying message sizes, test integrity of message using MD-5, SHA-1, and analyse the performance of the two protocols. Use crypt APIs.

Output :

```
users@Tauseef:~$ touch experi3.txt

users@Tauseef:~$ echo "Tauseef" >> experi3.txt

users@Tauseef:~$ md5sum experi3.txt
d1e87d22dd21e5a63a7578e055e750f8  experi3.txt

users@Tauseef:~$ sha1sum experi3.txt
2dd4a57416b419a354798122e23e34282435dc43  experi3.txt

users@Tauseef:~$ md5sum experi3.txt > experi3.md5 && md5sum -c experi3.md5
experi3.txt: OK

users@Tauseef:~$ sha1sum experi3.txt > experi3.sha1 && sha1sum -c experi3.sha1
experi3.txt: OK

users@Tauseef:~$ echo "Pathan" >> experi3.txt

users@Tauseef:~$ md5sum -c experi3.md5
experi3.txt: FAILED
md5sum: WARNING: 1 computed checksum did NOT match

users@Tauseef:~$ sha1sum -c experi3.sha1
experi3.txt: FAILED
sha1sum: WARNING: 1 computed checksum did NOT match
```

Conclusion:

Tested message integrity using MD-5, SHA-1, and analyzed the performance of the two protocols using crypt APIs for varying message sizes.

Course Code : CSL604 | **Course Name : System Security Lab**

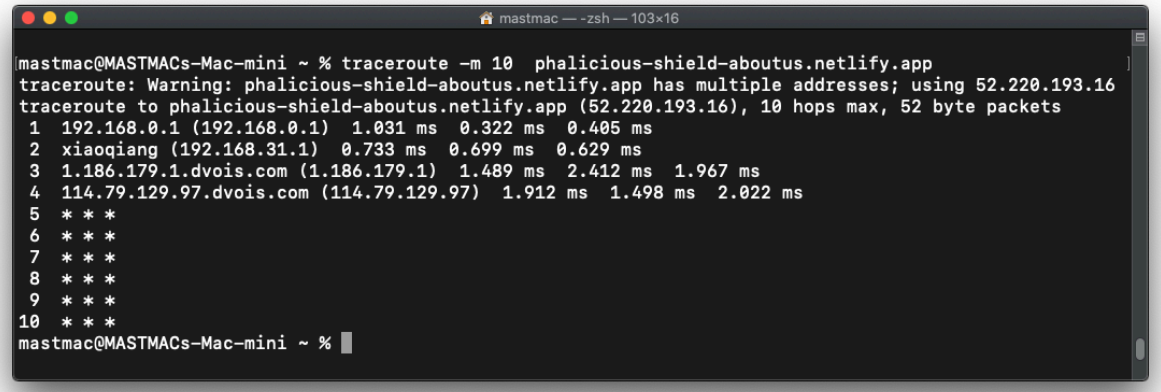
Class : TE-CO | **Batch : Computer Engineering**

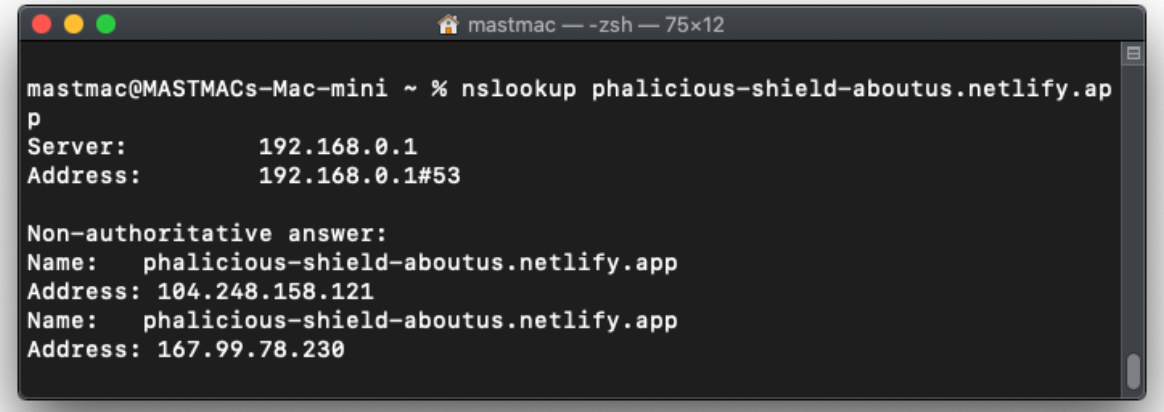
Roll no : 18CO63 | **Name : SHAIKH TAUSEEF MUSHTAQUE ALI**

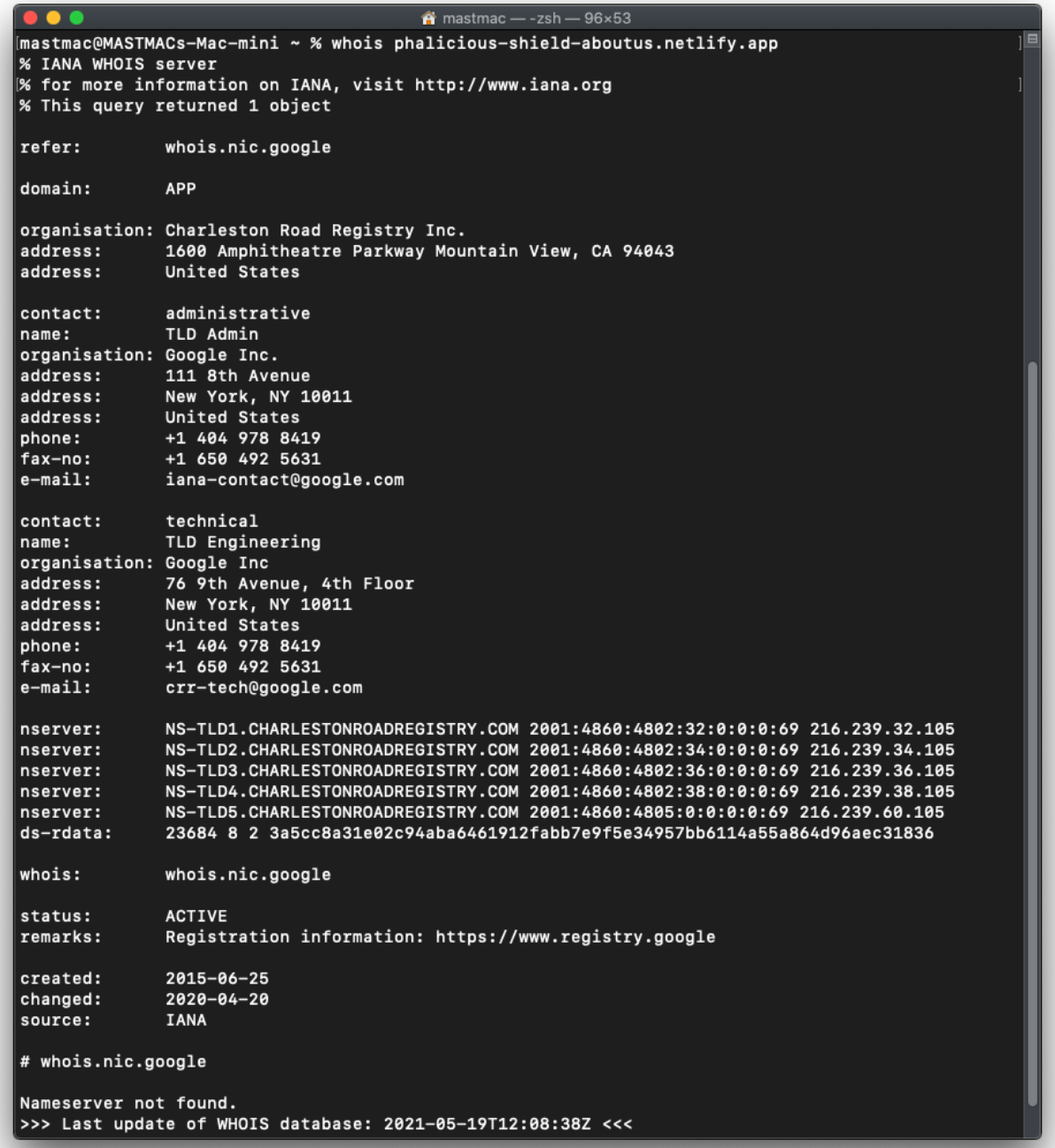
Experiment: 05

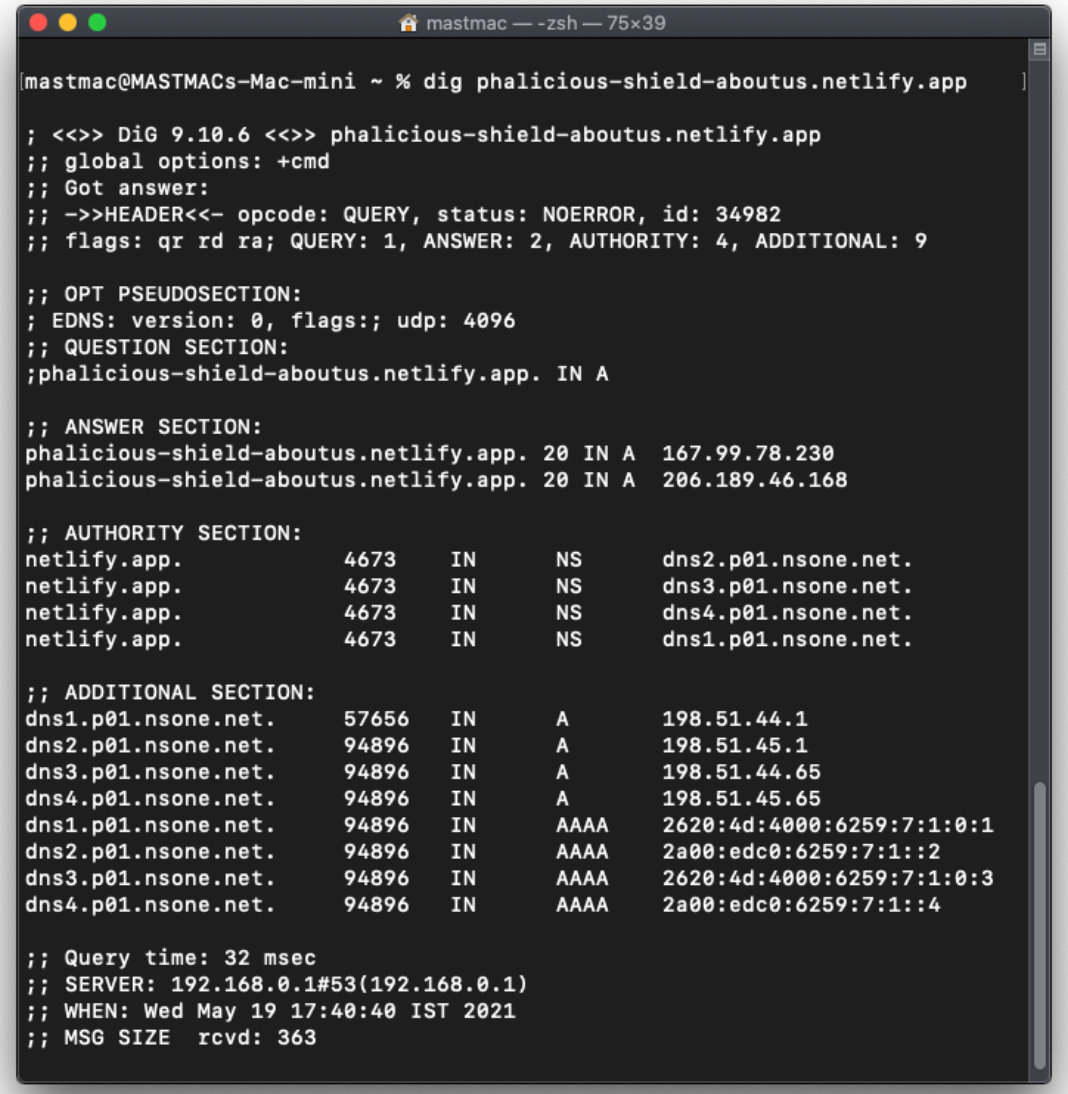
Aim : Study the use of network reconnaissance tools like WHOIS, dig, traceroute, nslookup to gather information about networks and domain registrars.

Output:






mastmac — -zsh — 96×53
mastmac@MASTMACs-Mac-mini ~ % whois phalicious-shield-aboutus.netlify.app
% IANA WHOIS server
% for more information on IANA, visit http://www.iana.org
% This query returned 1 object
refer: whois.nic.google
domain: APP
organisation: Charleston Road Registry Inc.
address: 1600 Amphitheatre Parkway Mountain View, CA 94043
address: United States
contact: administrative
name: TLD Admin
organisation: Google Inc.
address: 111 8th Avenue
address: New York, NY 10011
address: United States
phone: +1 404 978 8419
fax-no: +1 650 492 5631
e-mail: iana-contact@google.com
contact: technical
name: TLD Engineering
organisation: Google Inc
address: 76 9th Avenue, 4th Floor
address: New York, NY 10011
address: United States
phone: +1 404 978 8419
fax-no: +1 650 492 5631
e-mail: crr-tech@google.com
nserver: NS-TLD1.CHARLESTONROADREGISTRY.COM 2001:4860:4802:32:0:0:0:69 216.239.32.105
nserver: NS-TLD2.CHARLESTONROADREGISTRY.COM 2001:4860:4802:34:0:0:0:69 216.239.34.105
nserver: NS-TLD3.CHARLESTONROADREGISTRY.COM 2001:4860:4802:36:0:0:0:69 216.239.36.105
nserver: NS-TLD4.CHARLESTONROADREGISTRY.COM 2001:4860:4802:38:0:0:0:69 216.239.38.105
nserver: NS-TLD5.CHARLESTONROADREGISTRY.COM 2001:4860:4805:0:0:0:0:69 216.239.60.105
ds-rdata: 23684 8 2 3a5cc8a31e02c94aba6461912fabb7e9f5e34957bb6114a55a864d96aec31836
whois: whois.nic.google
status: ACTIVE
remarks: Registration information: https://www.registry.google
created: 2015-06-25
changed: 2020-04-20
source: IANA
whois.nic.google
Nameserver not found.
>>> Last update of WHOIS database: 2021-05-19T12:08:38Z <<<



Conclusion:

Network reconnaissance tools like WHOIS, dig, traceroute and nslookup are studied to gather information
about networks and domain registrars.

Course Code : CSL604 **Course Name : System Security Lab**

Class : TE-CO **Batch : Computer Engineering**

Roll no : 18CO63 **Name : SHAIKH TAUSEEF MUSHTAQUE ALI**

Experiment : 06

Aim : Study of packet sniffer tools : wireshark, 1. Download and install wireshark and capture icmp, tcp, and http packets in promiscuous mode.

Theory:

Introduction

The first part of the lab introduces packet sniffer, Wireshark. Wireshark is a free open- source network protocol analyzer. It is used for network troubleshooting and communication protocol analysis. Wireshark captures network packets in real time and display them in human-readable format. It provides many advanced features including live capture and offline analysis, three-pane packet browser, coloring rules for analysis. This document uses Wireshark for the experiments, and it covers Wireshark installation, packet capturing, and protocol analysis.

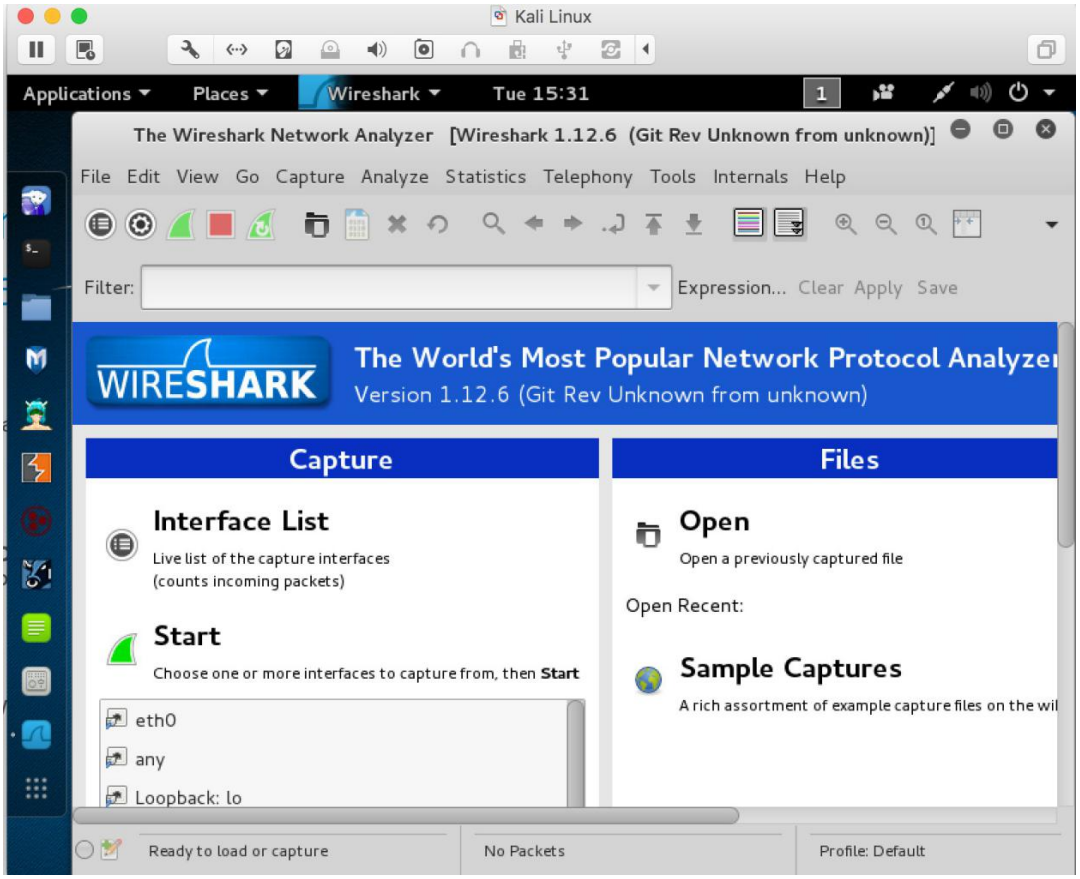


Figure 1: Wireshark in Kali Linux

Background

TCP/IP Network Stack

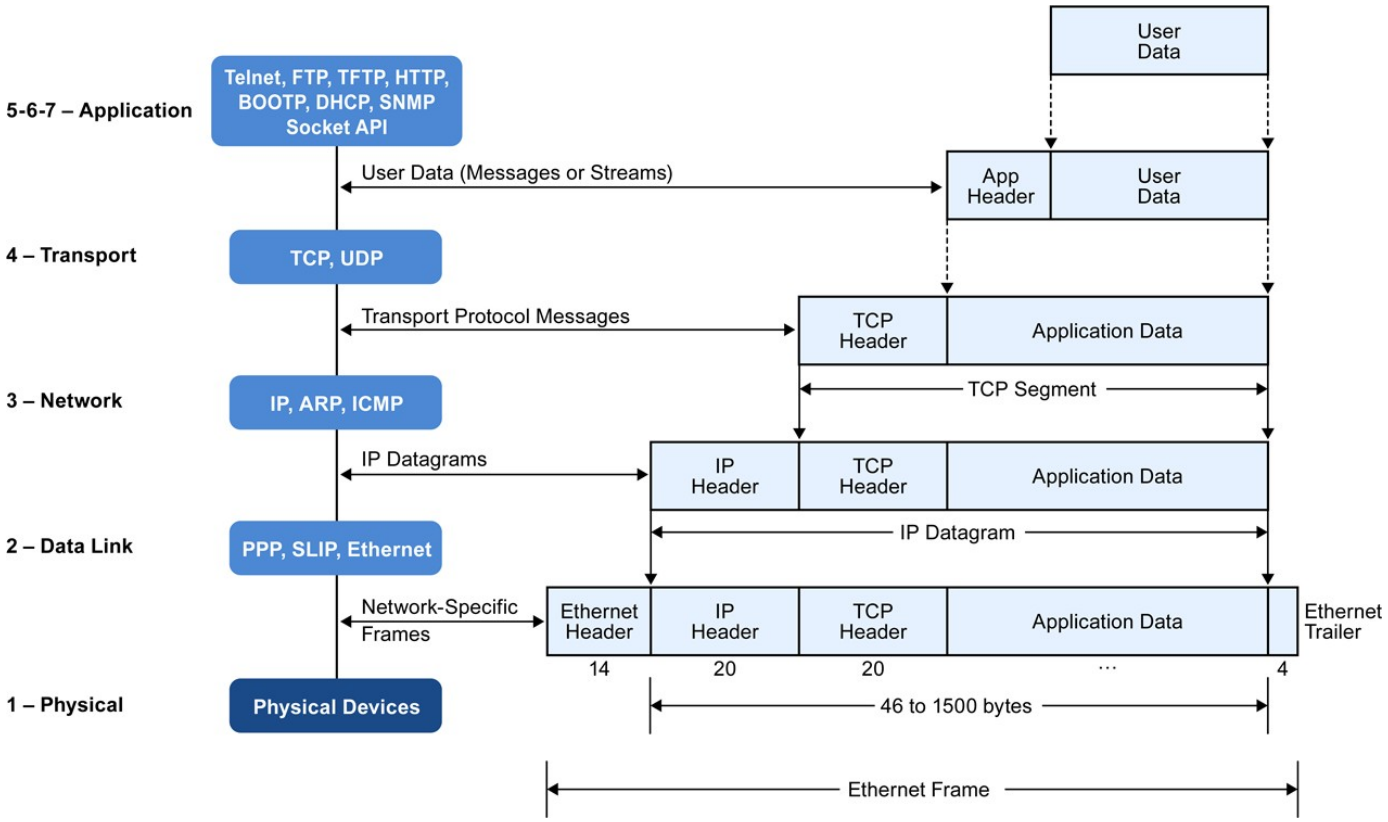


Figure 2: Encapsulation of Data in the TCP/IP Network Stack

In the CSC 4190 Introduction to Computer Networking, TCP/IP network stack is introduced and studied. This background section briefly explains the concept of TCP/IP network stack to help you better understand the experiments. TCP/IP is the most commonly used network model for Internet services. Because its most important protocols, the Transmission Control Protocol (TCP) and the Internet Protocol (IP) were the first networking protocols defined in this standard, it is named as TCP/IP. However, it contains multiple layers including application layer, transport layer, network layer, and data link layer.

- *Application Layer*: The application layer includes the protocols used by most applications for providing user services. Examples of application layer protocols are Hypertext Transfer Protocol (HTTP), Secure Shell (SSH), File Transfer Protocol (FTP), and Simple Mail Transfer Protocol (SMTP).
- *Transport Layer*: The transport layer establishes process-to-process connectivity, and it provides end-to-end services that are independent of underlying user data. To implement the process-to-process communication, the protocol introduces a concept of port. The examples of transport layer protocols are Transport Control Protocol (TCP) and User Datagram Protocol (UDP). The TCP provides flow- control, connection establishment, and reliable transmission of data, while the UDP is a connectionless transmission model.
- *Internet Layer*: The Internet layer is responsible for sending packets to across networks. It has two functions: 1) Host identification by using IP addressing system (IPv4 and IPv6); and 2) packets routing

from source to destination. The examples of Internet layer protocols are Internet Protocol (IP), Internet Control Message Protocol (ICMP), and Address Resolution Protocol (ARP).

- *Link Layer*: The link layer defines the networking methods within the scope of the local network link. It is used to move the packets between two hosts on the same link. An common example of link layer protocols is Ethernet.

Packet Sniffer

Packet sniffer is a basic tool for observing network packet exchanges in a computer. As the name suggests, a packet sniffer captures ("sniffs") packets being sent/received from/by your computer; it will also typically store and/or display the contents of the various protocol fields in these captured packets. A packet sniffer itself is passive. It observes messages being sent and received by applications and protocols running on your computer, but never sends packets itself.

Figure 3 shows the structure of a packet sniffer. At the right of **Figure** 3 are the protocols (in this case, Internet protocols) and applications (such as a web browser or ftp client) that normally run on your computer. The packet sniffer, shown within the dashed rectangle in **Figure** 3 is an addition to the usual software in your computer, and consists of two parts. The packet capture library receives a copy of every link-layer frame that is sent from or received by your computer. Messages exchanged by higher layer protocols such as HTTP, FTP, TCP, UDP, DNS, or IP all are eventually encapsulated in link-layer frames that are transmitted over physical media such as an Ethernet cable. In Figure 1, the assumed physical media is an Ethernet, and so all upper-layer protocols are eventually encapsulated within an Ethernet frame. Capturing all link-layer frames thus gives you access to all messages sent/received from/by all protocols and applications executing in your computer.

The second component of a packet sniffer is the packet analyzer, which displays the contents of all fields within a protocol message. In order to do so, the packet analyzer.

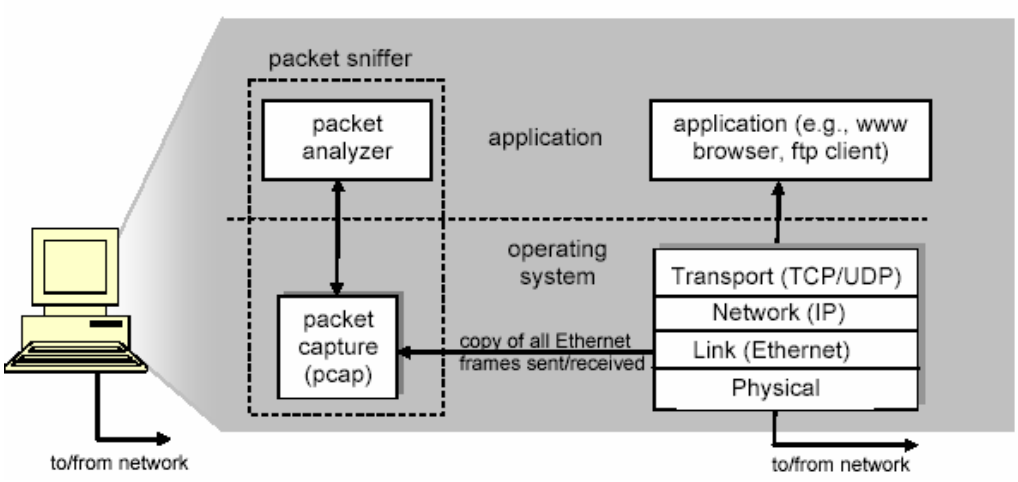


Figure 3: Packet Sniffer Structure

must "understand" the structure of all messages exchanged by protocols. For example, suppose we are interested in displaying the various fields in messages exchanged by the HTTP protocol in **Figure** 3. The packet analyzer understands the format of Ethernet frames, and so can identify the IP datagram within an Ethernet frame. It also understands the IP datagram format, so that it can extract the TCP segment within the IP datagram. Finally, it understands the TCP segment structure, so it can extract the HTTP message contained in the TCP segment. Finally,

it understands the HTTP protocol and so, for example, knows that the first bytes of an HTTP message will contain the string "GET," "POST," or "HEAD".

We will be using the Wireshark packet sniffer [http://www.wireshark.org/] for these labs, allowing us to display the contents of messages being sent/received from/by protocols at different levels of the protocol stack. (Technically speaking, Wireshark is a packet analyzer that uses a packet capture library in your computer). Wireshark is a free network protocol analyzer that runs on Windows, Linux/Unix, and Mac computers.

Getting Wireshark

The Kai Linux has Wireshark installed. You can just launch the Kali Linux VM and open Wireshark there. Wireshark can also be downloaded from here:

https://www.wireshark.org/download.html

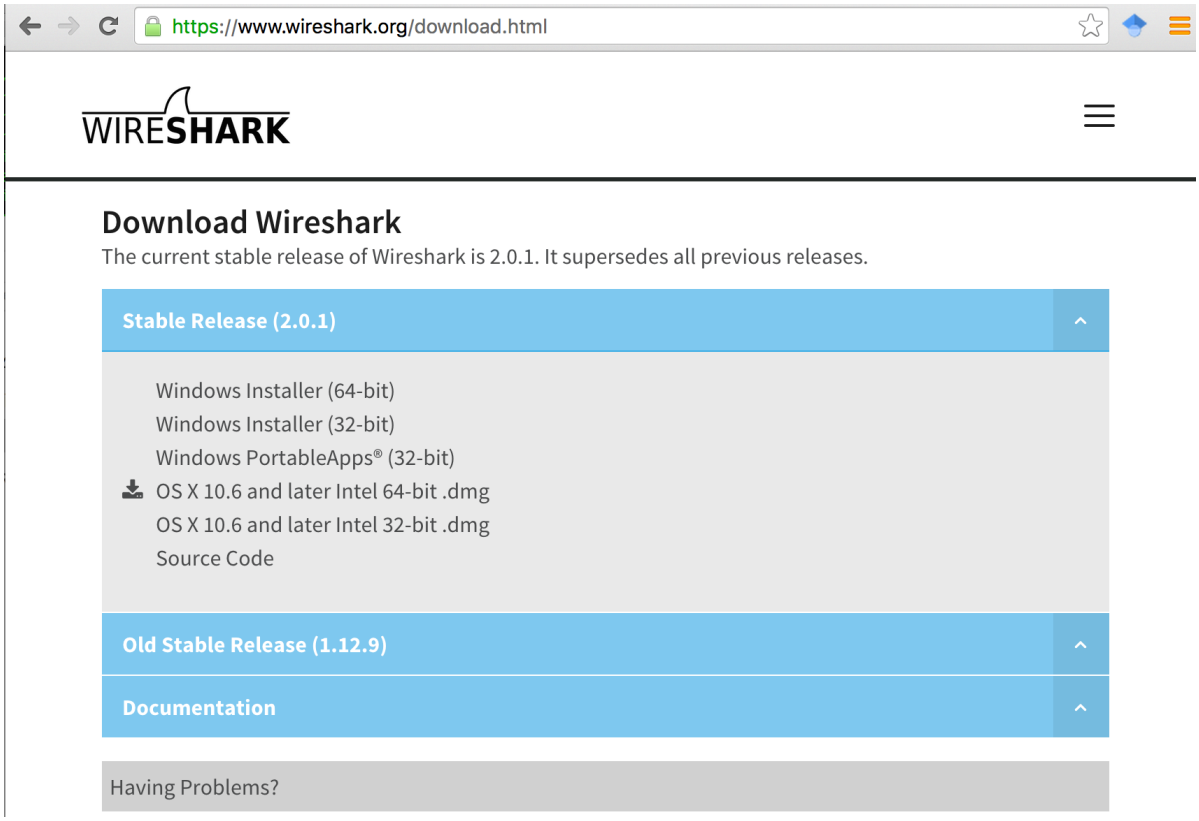


Figure 4: Download Page of Wireshark

Starting Wireshark

When you run the Wireshark program, the Wireshark graphic user interface will be shown as **Figure** 5. Currently, the program is not capturing the packets.

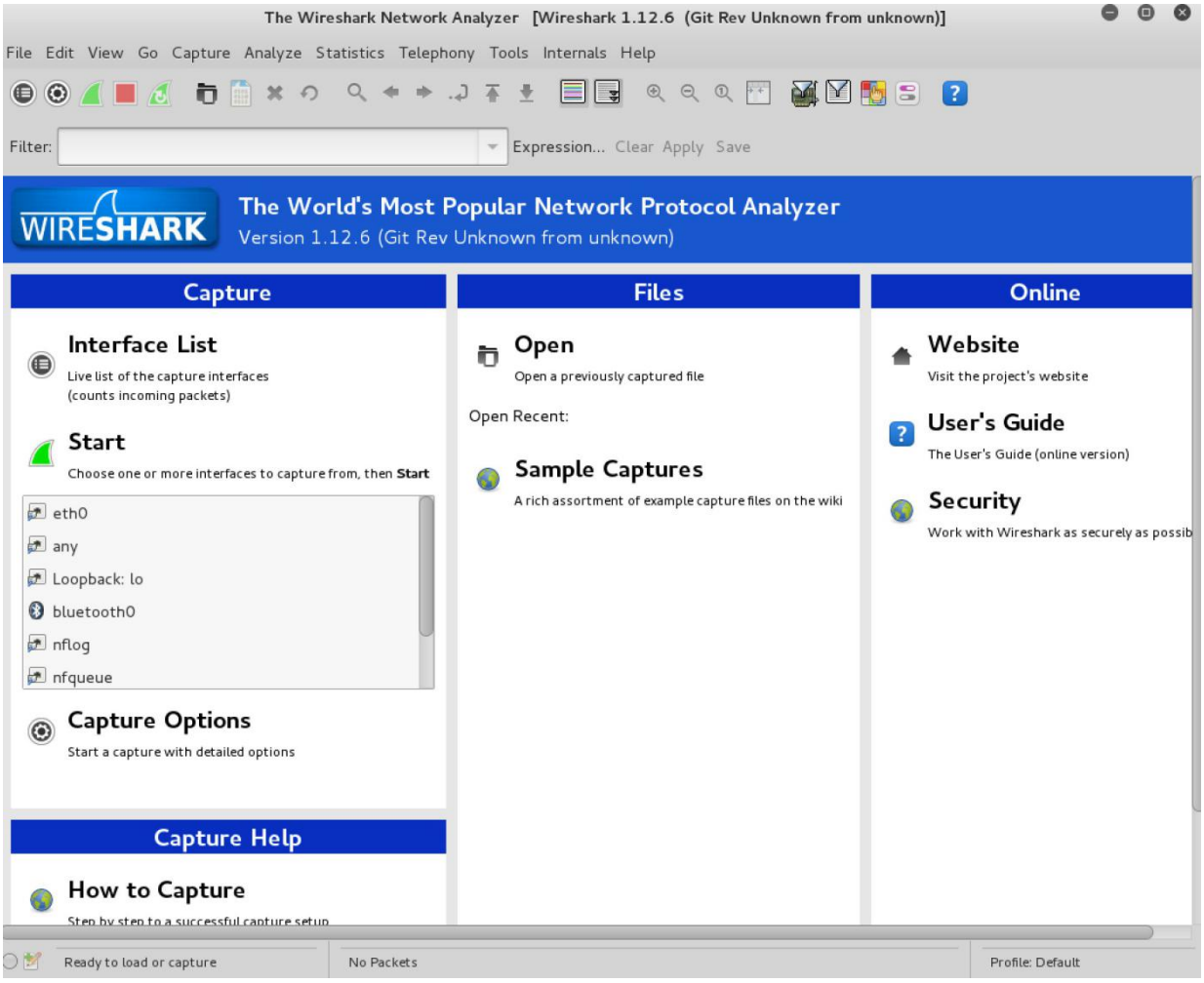


Figure 5: Initial Graphic User Interface of Wireshark

Then, you need to choose an interface. If you are running the Wireshark on your laptop, you need to select WiFi interface. If you are at a desktop, you need to select the Ethernet interface being used. Note that there could be multiple interfaces. In general, you can select any interface but that does not mean that traffic will flow through that interface. The network interfaces (i.e., the physical connections) that your computer has to the network are shown. The attached **Figure** 6 was taken from my computer.

After you select the interface, you can click start to capture the packets as shown in

Figure 7.

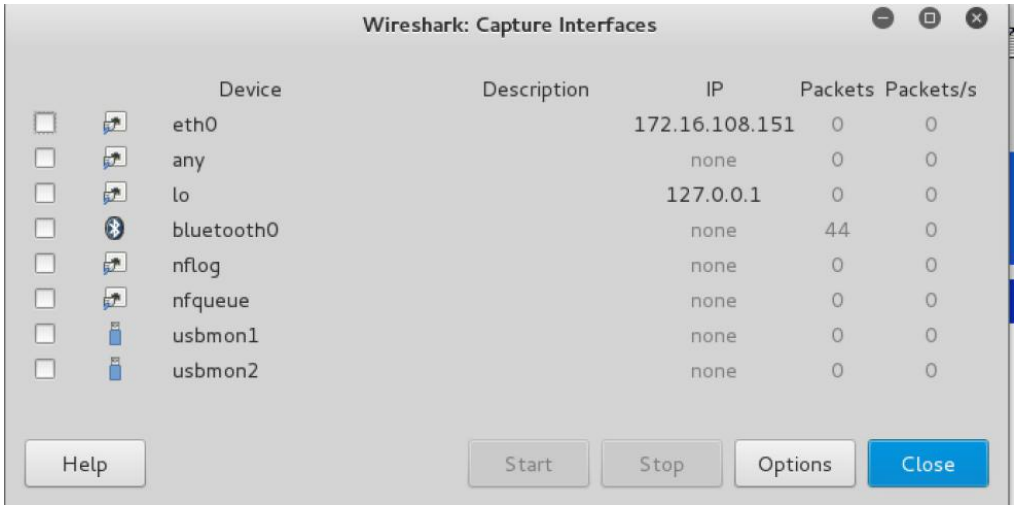


Figure 6: Capture Interfaces in Wireshark

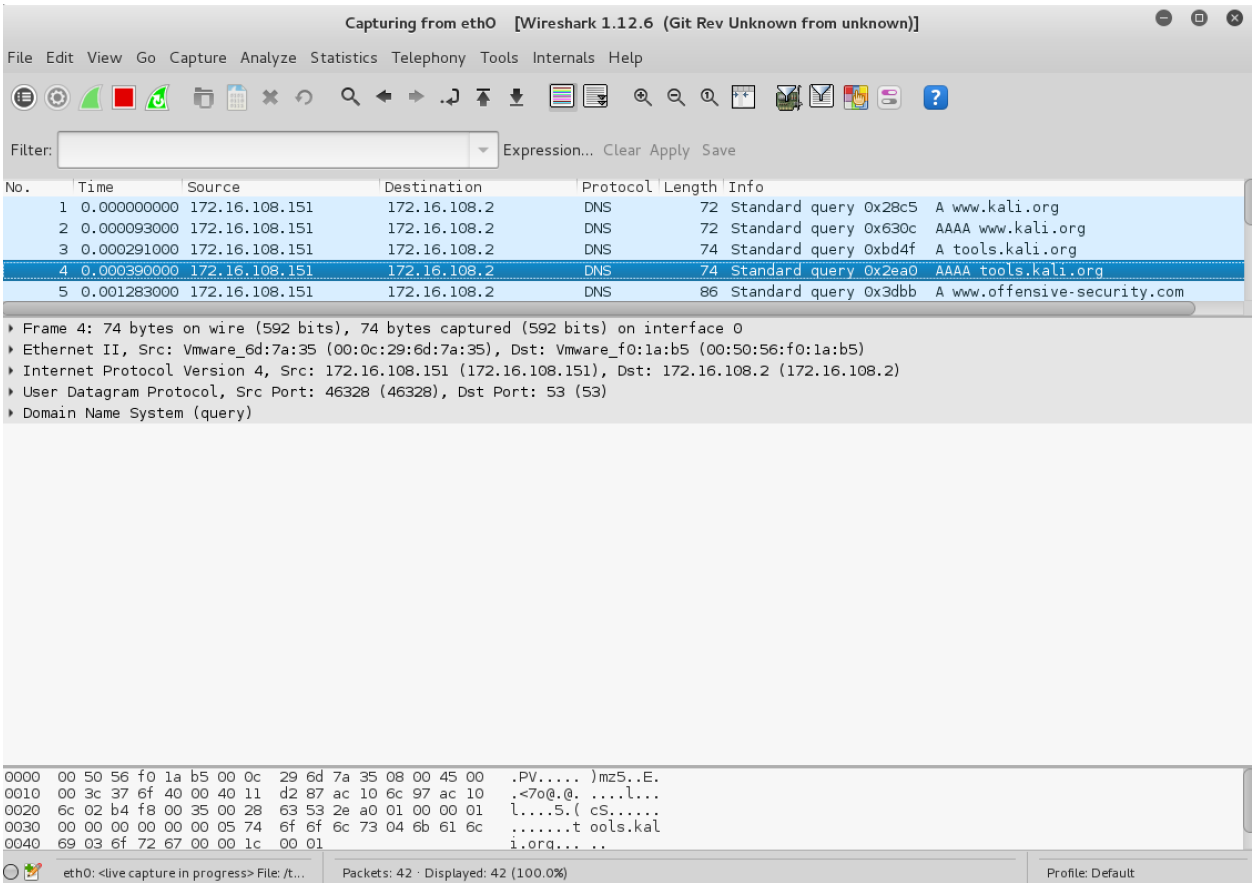


Figure 7: Capturing Packets in Wireshark

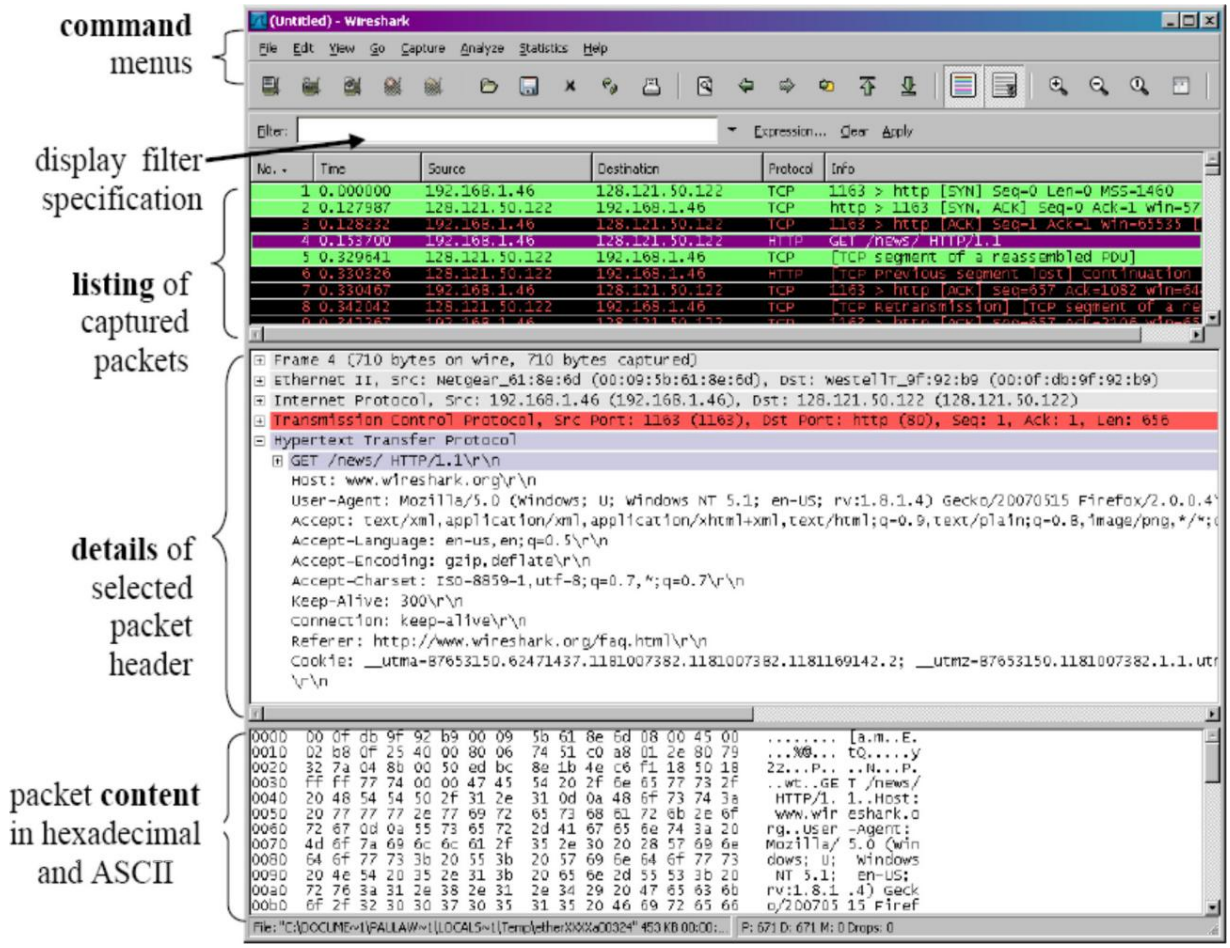


Figure 8: Wireshark Graphical User Interface on Microsoft Windows

The Wireshark interface has five major components:

The **command menus** are standard pulldown menus located at the top of the window. Of interest to us now is the File and Capture menus. The File menu allows you to save captured packet data or open a file containing previously captured packet data, and exit the Wireshark application. The Capture menu allows you to begin packet capture.

The **packet-listing window** displays a one-line summary for each packet captured, including the packet number (assigned by Wireshark; this is not a packet number contained in any protocol's header), the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing can be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest- level protocol that sent or received this packet, i.e., the protocol that is the source or ultimate sink for this packet.

The **packet-header details window** provides details about the packet selected (highlighted) in the packet-listing window. (To select a packet in the packet-listing window, place the cursor over the packet's one-line summary in the packet-listing window and click with the left mouse button.). These details include information about the Ethernet frame and IP datagram that contains this packet. The amount of Ethernet and IP-layer detail displayed can be expanded or minimized by clicking on the right- pointing or down-pointing arrowhead to the left of the Ethernet

frame or IP datagram line in the packet details window. If the packet has been carried over TCP or UDP, TCP or UDP details will also be displayed, which can similarly be expanded or minimized. Finally, details about the highest-level protocol that sent or received this packet are also provided.

The **packet-contents window** displays the entire contents of the captured frame, in both ASCII and hexadecimal format.

Towards the top of the Wireshark graphical user interface, is the **packet display filter field**, into which a protocol name or other information can be entered in order to filter the information displayed in the packet-listing window (and hence the packet-header and packet-contents windows). In the example below, we'll use the packet-display filter field to have Wireshark hide (not display) packets except those that correspond to HTTP messages.

Capturing Packets

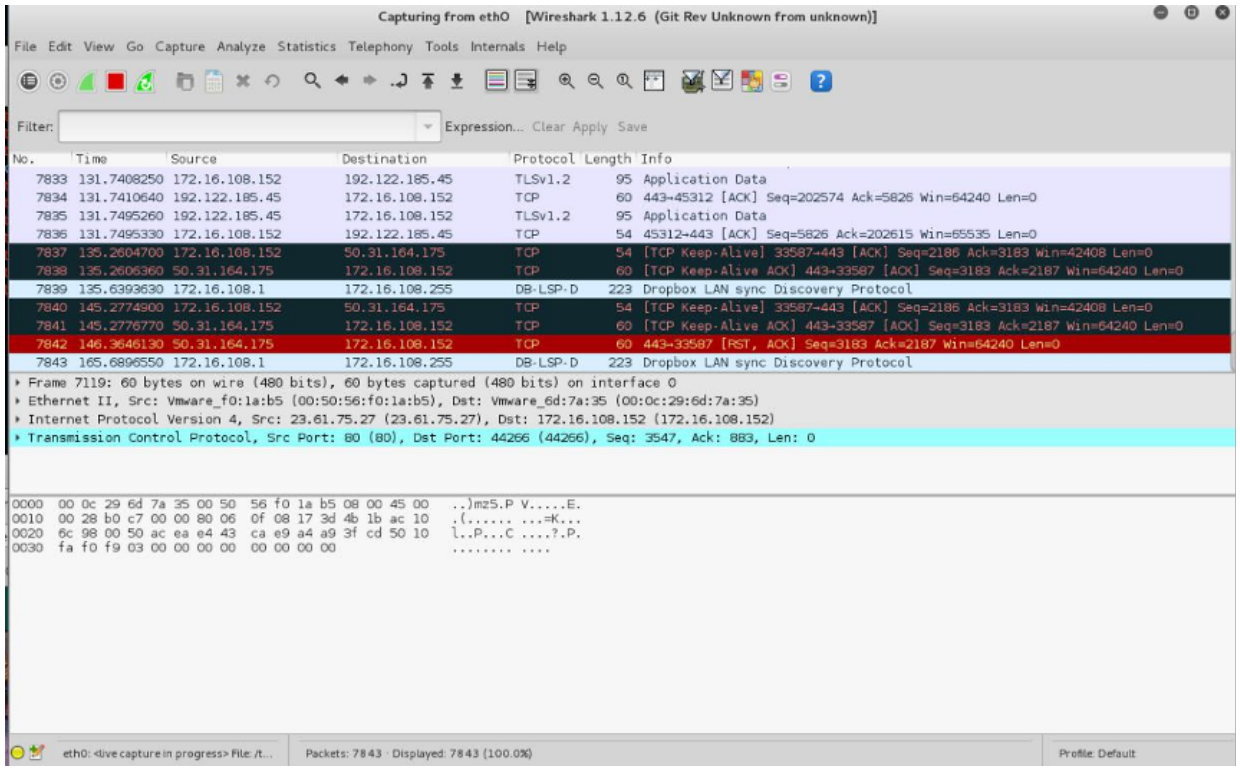
After downloading and installing Wireshark, you can launch it and click the name of an interface under Interface List to start capturing packets on that interface. For example, if you want to capture traffic on the wireless network, click your wireless interface.

Test Run

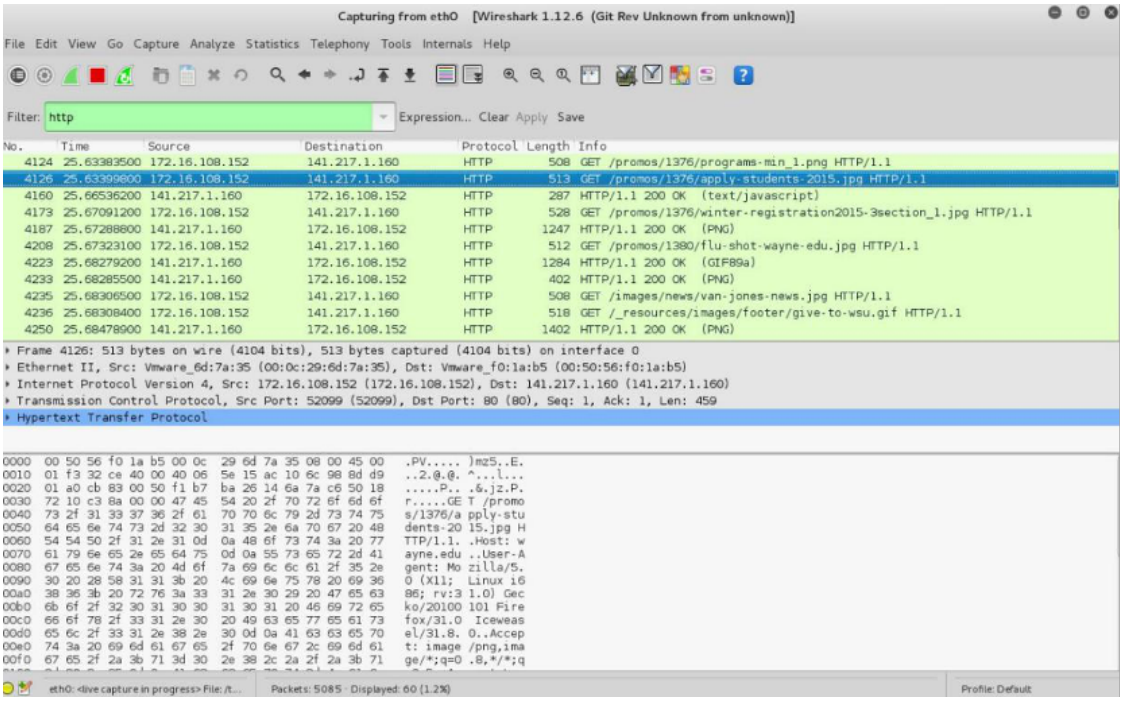
Do the following steps:

1. Start up the Wireshark program (select an interface and press start to capture packets).
2. Start up your favorite browser (ceweasel in Kali Linux).
3. In your browser, go to Wayne State homepage by typing www.wayne.edu.
4. After your browser has displayed the http://www.wayne.edu page, stop Wireshark packet capture by selecting stop in the Wireshark capture window. This will cause the Wireshark capture window to disappear and the main Wireshark window to display all packets captured since you began packet capture see image below:

5. Color Coding: You'll probably see packets highlighted in green, blue, and black. Wireshark uses colors to help you identify the types of traffic at a glance. By default, green is TCP traffic, dark blue is DNS traffic, light blue is UDP traffic, and black identifies TCP packets with problems — for example, they could have been delivered out-of-order.

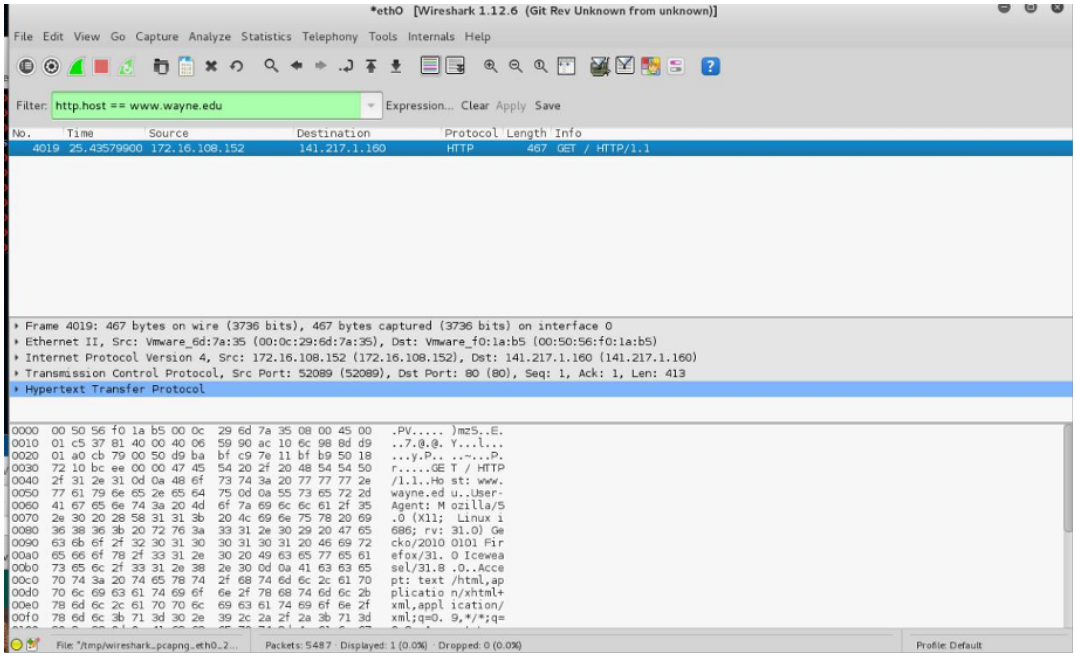

Capturing from eth0 [Wireshark 1.12.6 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: Expression... Clear Apply Save
No. Time Source Destination Protocol Length Info
7833 131.7408250 172.16.108.152 192.122.185.45 TLSv1.2 95 Application Data
7834 131.7410640 192.122.185.45 172.16.108.152 TCP 60 443→45312 [ACK] Seq=202574 Ack=5826 Win=64240 Len=0
7835 131.7495260 192.122.185.45 172.16.108.152 TLSv1.2 95 Application Data
7836 131.7495330 172.16.108.152 192.122.185.45 TCP 54 45312→443 [ACK] Seq=5826 Ack=202615 Win=65535 Len=0
7837 135.2604700 172.16.108.152 50.31.164.175 TCP 54 [TCP Keep-Alive] 33587→443 [ACK] Seq=2186 Ack=3183 Win=42408 Len=0
7838 135.2606360 50.31.164.175 172.16.108.152 TCP 60 [TCP Keep-Alive ACK] 443→33587 [ACK] Seq=3183 Ack=2187 Win=64240 Len=0
7839 135.6393630 172.16.108.1 172.16.108.255 DB-LSP-D 223 Dropbox LAN sync Discovery Protocol
7840 145.2774900 172.16.108.152 50.31.164.175 TCP 54 [TCP Keep-Alive] 33587→443 [ACK] Seq=2186 Ack=3183 Win=42408 Len=0
7841 145.2776770 50.31.164.175 172.16.108.152 TCP 60 [TCP Keep-Alive ACK] 443→33587 [ACK] Seq=3183 Ack=2187 Win=64240 Len=0
7842 146.3646130 50.31.164.175 172.16.108.152 TCP 60 443→33587 [RST, ACK] Seq=3183 Ack=2187 Win=64240 Len=0
7843 165.6896550 172.16.108.1 172.16.108.255 DB-LSP-D 223 Dropbox LAN sync Discovery Protocol
Frame 7119: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
Ethernet II, Src: Vmware_f0:1a:b5 (00:50:56:f0:1a:b5), Dst: Vmware_6d:7a:35 (00:0c:29:6d:7a:35)
Internet Protocol Version 4, Src: 23.61.75.27 (23.61.75.27), Dst: 172.16.108.152 (172.16.108.152)
Transmission Control Protocol, Src Port: 80 (80), Dst Port: 44266 (44266), Seq: 3547, Ack: 883, Len: 0
0000 00 0c 29 6d 7a 35 00 50 56 f0 1a b5 08 00 45 00 ..)mz5.P V.....E.
0010 00 28 b0 c7 00 00 80 06 0f 08 17 3d 4b 1b ac 10 .(..... ...=K...
0020 6c 98 00 50 ac ea e4 43 ca e9 a4 a9 3f cd 50 10 l..P...C ....?.P.
0030 fa f0 f9 03 00 00 00 00 00 00 00 00 ....
eth0: <live capture in progress> File: /t...
Packets: 7843 · Displayed: 7843 (100.0%)
Profile: Default

6. You now have live packet data that contains all protocol messages exchanged between your computer and other network entities! However, as you will notice the HTTP messages are not clearly shown because there are many other packets included in the packet capture. Even though the only action you took was to open your browser, there are many other programs in your computer that communicate via the network in the background. To filter the connections to the ones we want to focus on, we have to use the filtering functionality of Wireshark by typing "http" in the filtering field as shown below:

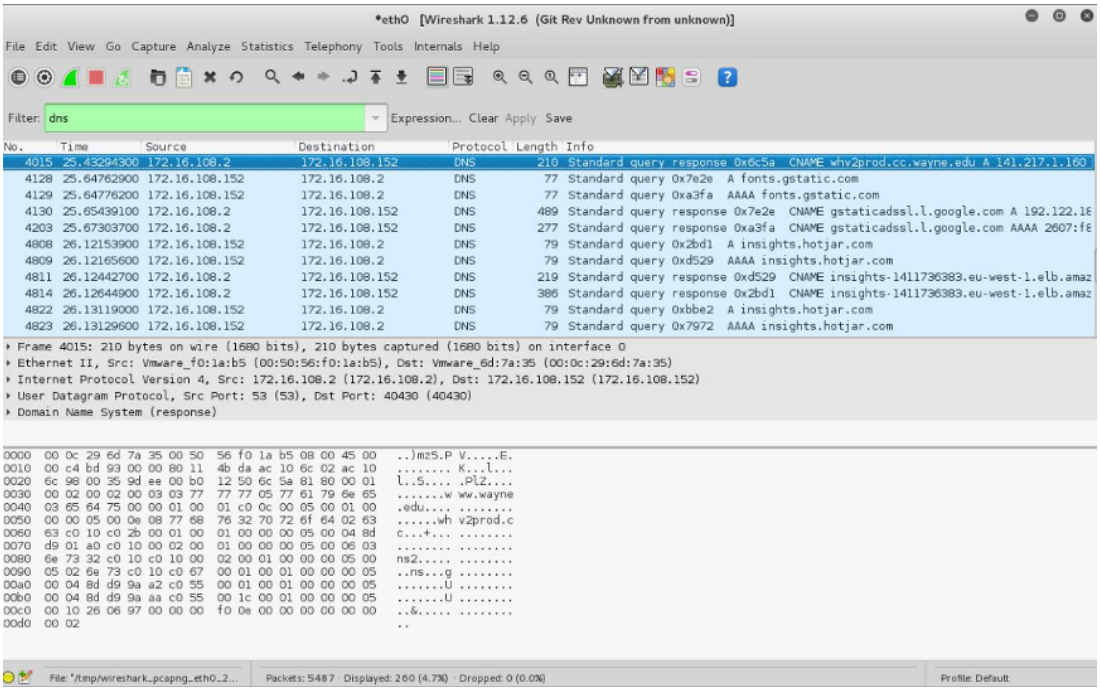


Notice that we now view only the packets that are of protocol HTTP. However, we also still do not have the exact communication we want to focus on because using HTTP as a filter is not descriptive enough to allow us to find our connection to http://www.wayne.edu. We need to be more precise if we want to capture the correct set of packets.

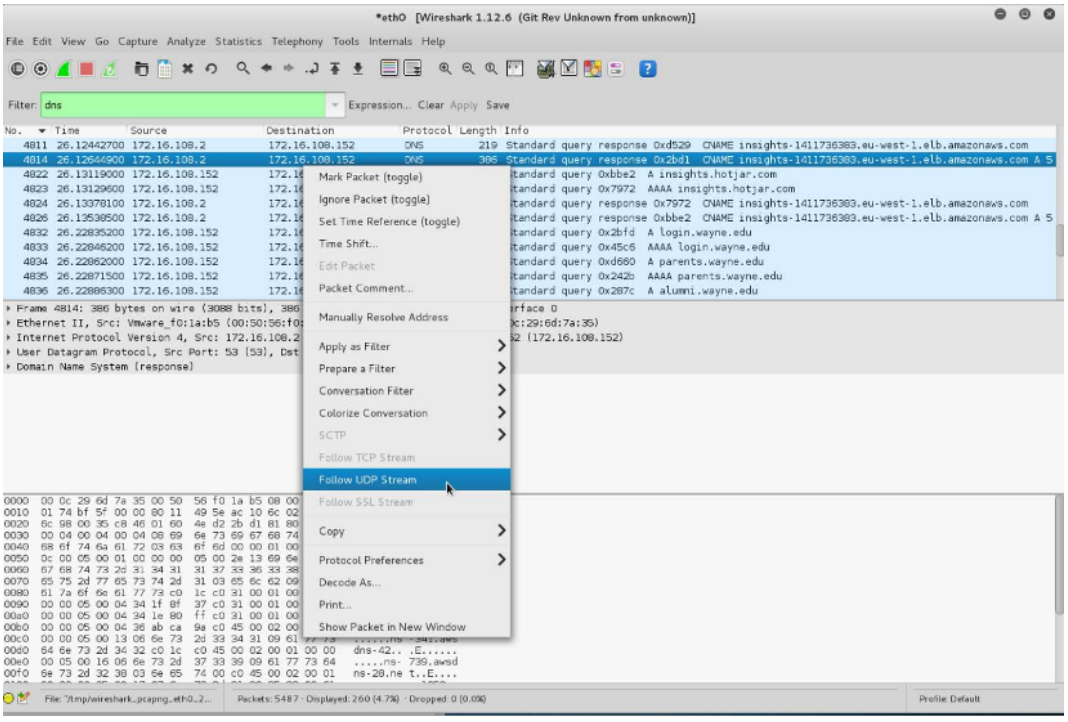
7. To further filter packets in Wireshark, we need to use a more precise filter. By setting the http.host==www.wayne.edu, we are restricting the view to packets that have as an http host the www.wayne.edu website. Notice that we need two equal signs to perform the match "==" not just one. See the screenshot below:



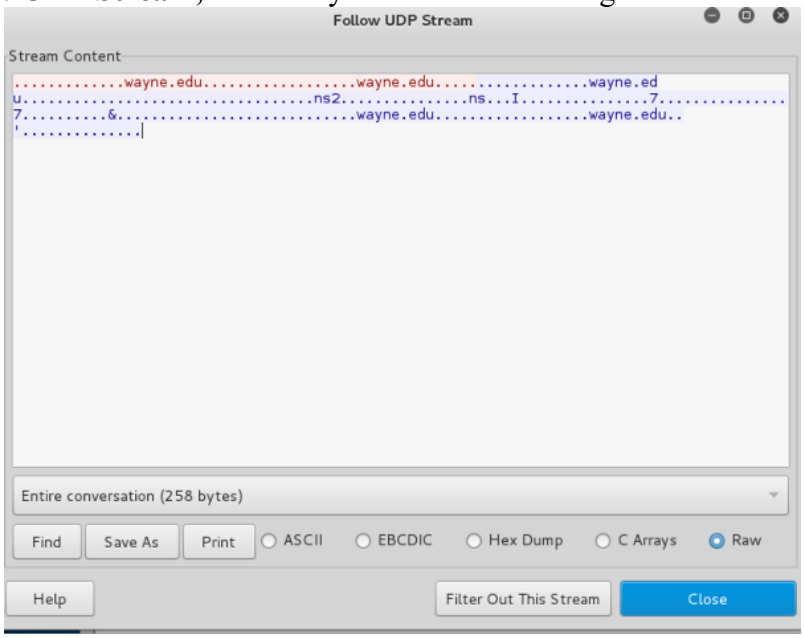
8. Now, we can try another protocol. Let's use Domain Name System (DNS) protocol as an example here.



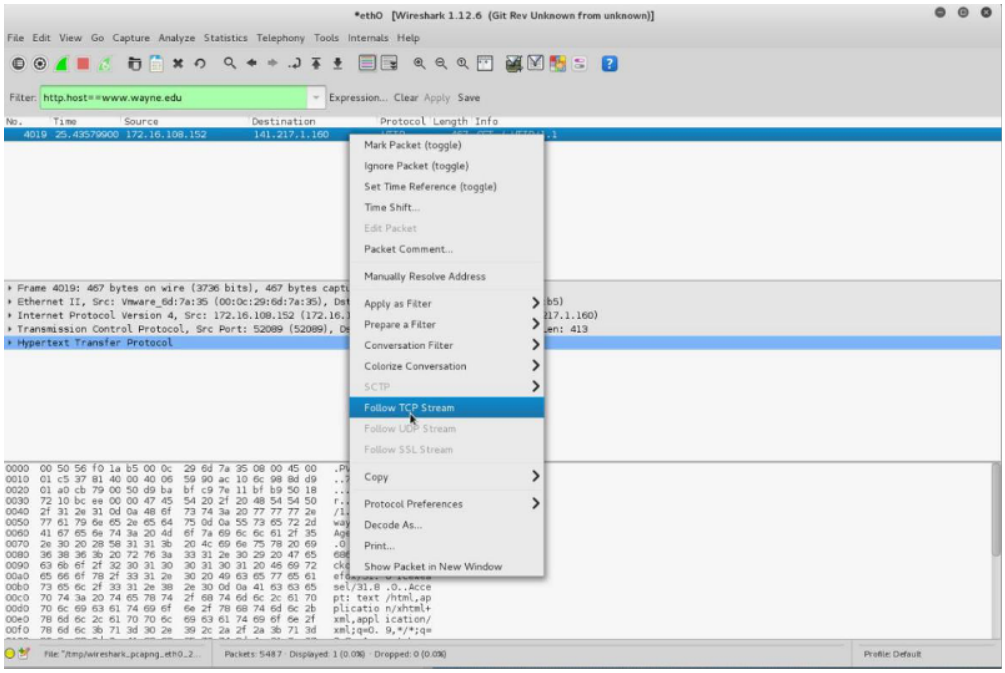
9. Let's try now to find out what are those packets contain by following one of the conversations (also called network flows), select one of the packets and press the right mouse button (if you are on a Mac use the command button and click), you should see something similar to the screen below:

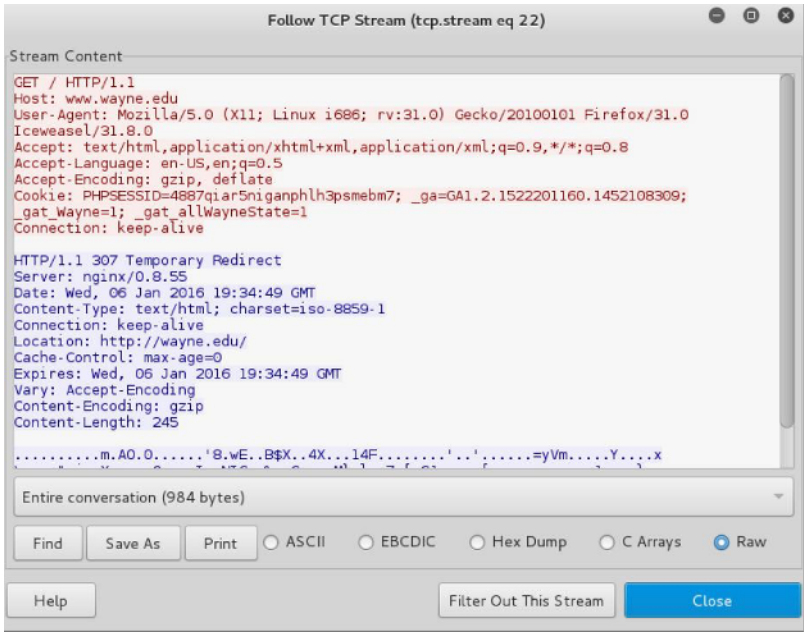


Click on **Follow UDP Stream,** and then you will see following screen.



10. If we close this window and change the filter back to "http.host==www.wayne.edu" and then follow a packet from the list of packets that match that filter, we should get the something similar to the following screens. Note that we click on **Follow TCP Stream** this time.





Conclusion : Captured icmp, tcp and http packets in promiscuous mode by packet sniffer tool wireshark.

Course Code : CSL604 **Course Name : System Security Lab**

Class : TE-CO **Batch : Computer Engineering**

Roll no : 18CO63 **Name : SHAIKH TAUSEEF MUSHTAQUE ALI**

Experiment : 07

Aim : Download and install nmap. Use it with different options to scan open ports, perform OS fingerprinting, do a ping scan, tcp port scan, udp port scan, xmas scan etc.

Theory :

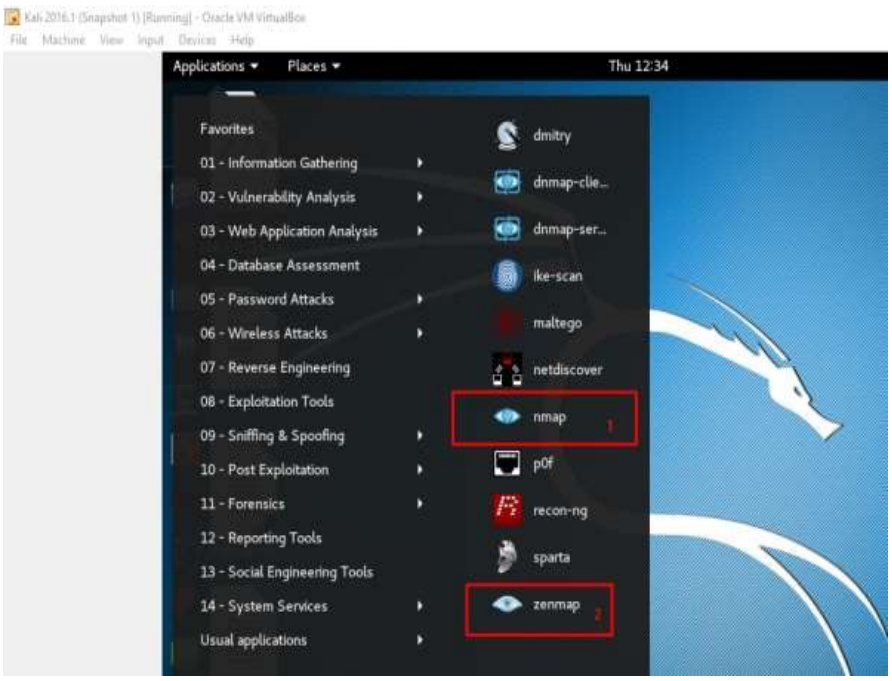
NMAPandZenMAP

NMAP and ZenMAP are useful tools for the scanning phase of Ethical Hacking in Kali Linux. NMAP and ZenMAP are practically the same tool, however NMAP uses command line while ZenMAP has a GUI.

NMAP is a free utility tool for network discovery and security auditing. Many systems and network administrators also find it useful for tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime.

NMAP uses raw IP packets in novel ways to determine which hosts are available on the network, what services (application name and version) those hosts are offering, which operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, etc.

Now, let's go step by step and learn how to use NMAP and ZenMAP.



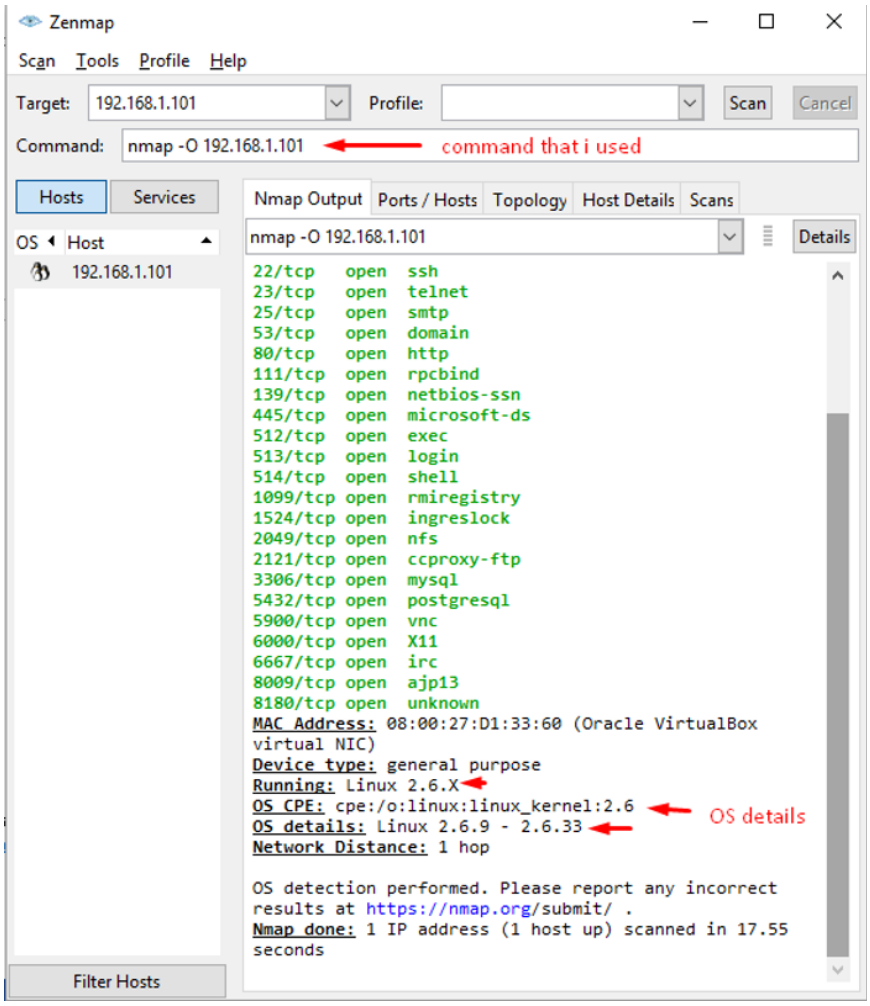
Step 1: To open, go to Applications -> 01-Information Gathering -> nmap or zenmap.

Step 2: The next step is to detect the OS type/version of the target host. Based on the help indicated by NMAP, the parameter of OS type/version detection is variable "-O". For more information, use this link: https://nmap.org/book/man-os-detection.html

The command that we will use is:

```
nmap –O 192.168.1.101
```

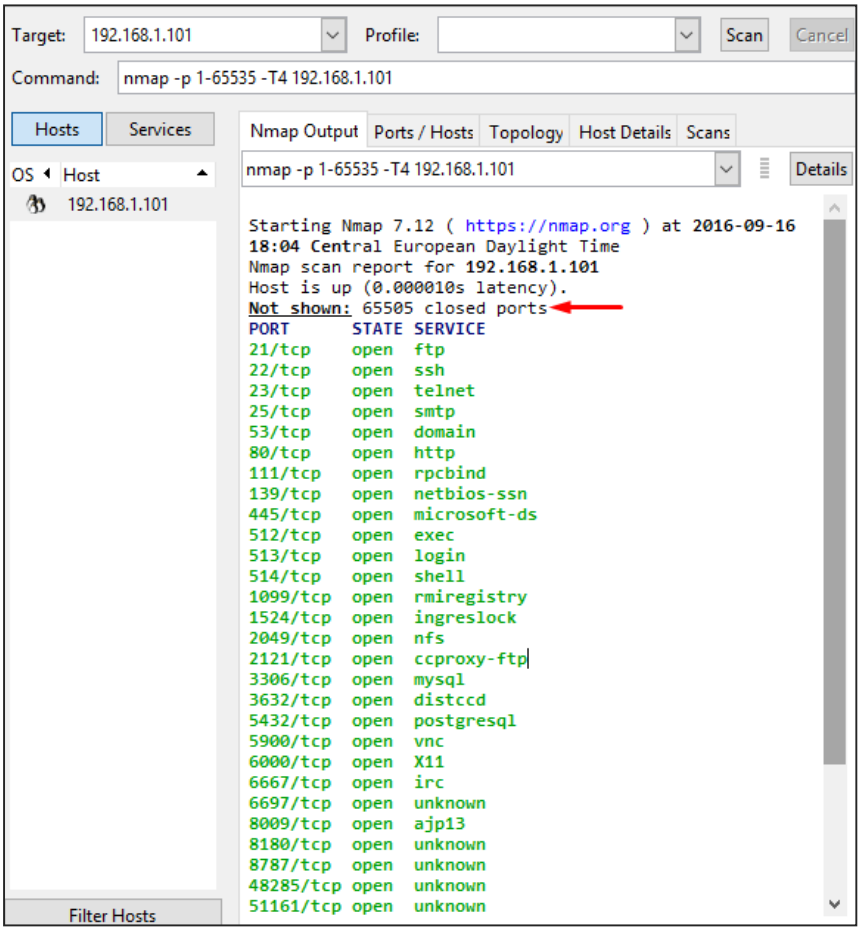
The following screenshot shows where you need to type the above command to see the Nmap output:



Step 3: Next, open the TCP and UDP ports. To scan all the TCP ports based on NMAP, use the following command:

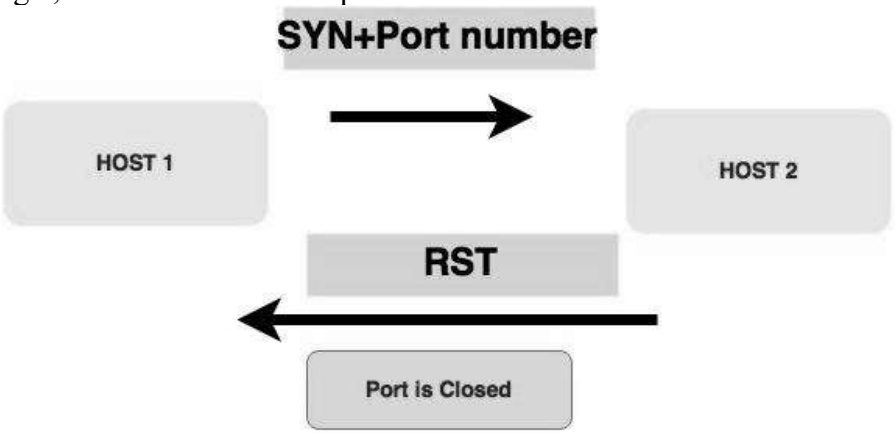
```
nmap –p 1–65535 –T4  192.168.1.101
```

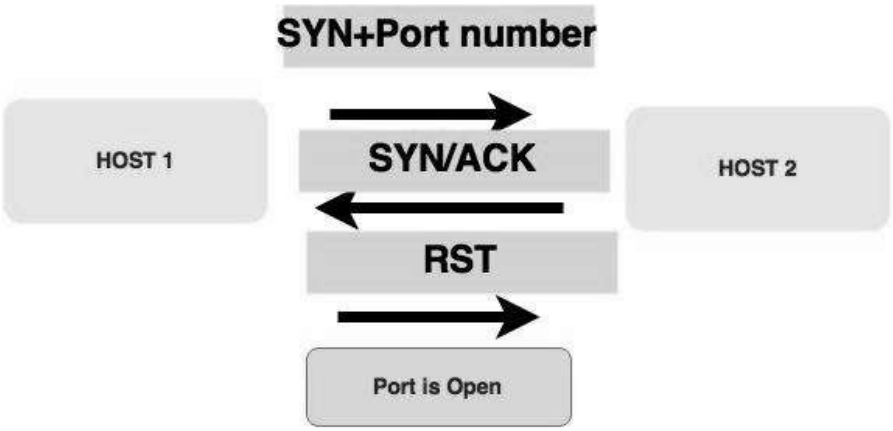
Where the parameter "–p" indicates all the TCP ports that have to be scanned. In this case, we are scanning all the ports and "-T4" is the speed of scanning at which NMAP has to run.
Following are the results. In green are all the TCP open ports and in red are all the closed ports. However, NMAP does not show as the list is too long.



StealthScan

Stealth scan or SYN is also known as **half-open scan**, as it doesn't complete the TCP three-way handshake. A hacker sends a SYN packet to the target; if a SYN/ACK frame is received back, then it's assumed the target would complete the connect and the port is listening. If an RST is received back from the target, then it is assumed the port isn't active or is closed.

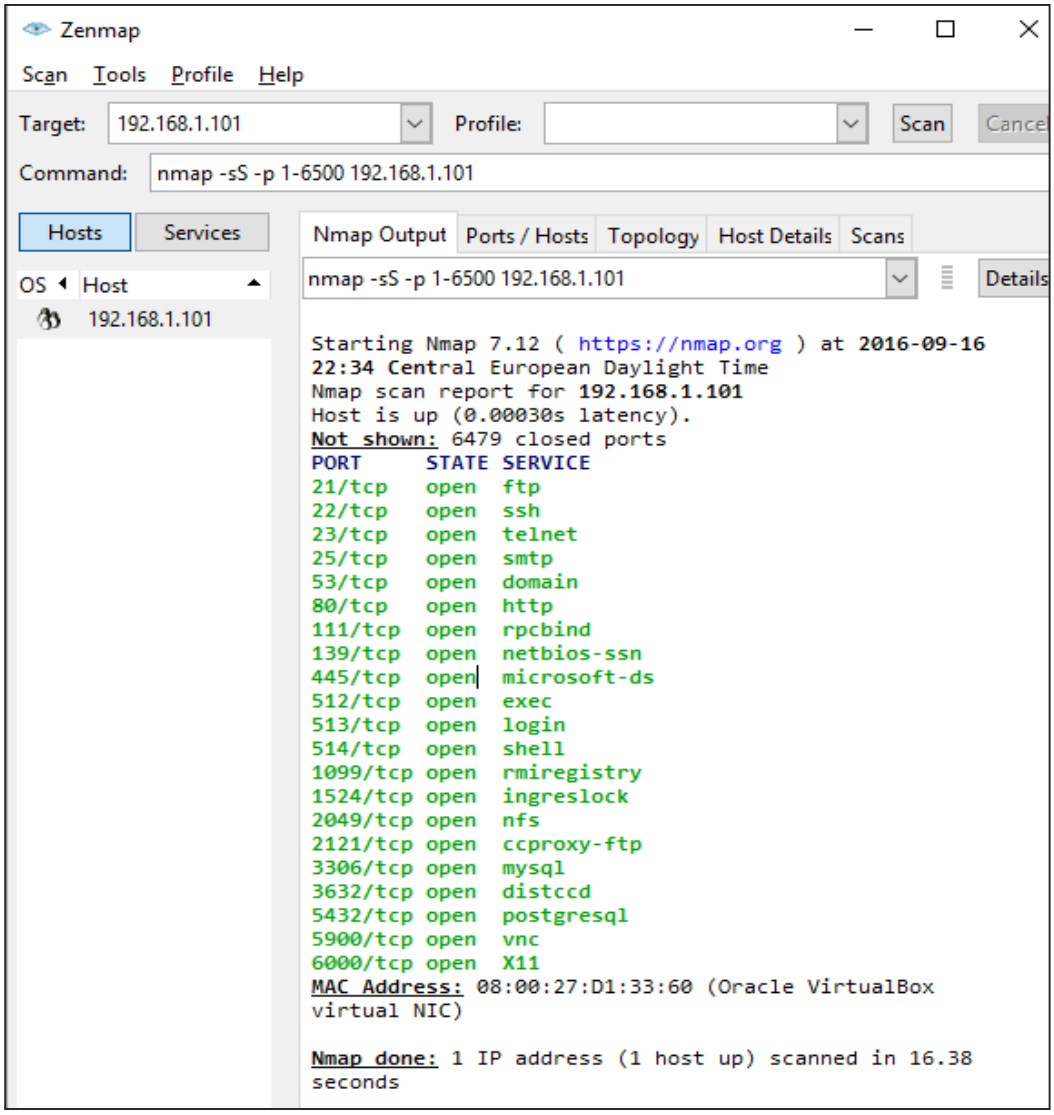




Now to see the SYN scan in practice, use the parameter **–sS** in NMAP. Following is the full command –

```
nmap –sS –T4 192.168.1.101
```

The following screenshot shows how to use this command:



Conclusion:

Used different options to scan open ports, perform OS fingerprinting, do a ping scan, tcp port scan, udp port
scan, xmas scan etc. by downloading nmap.

Course Code : CSL604 **Course Name : System Security Lab**

Class : TE-CO **Batch : Computer Engineering**

Roll no : 18CO63 **Name : SHAIKH TAUSEEF MUSHTAQUE ALI**

Experiment : 08

Aim : Setting up personal Firewall using iptables.

Theory :

iptables is a firewall program for **Linux**. It will monitor traffic from and to your server using **tables**. These tables contain **sets of rules**, called **chains**, that will filter incoming and outgoing data packets.

[optin-monster-shortcode id="fv4lqeko3gylvecpszws"]

When a **packet** matches a **rule**, it is given a **target**, which can be another chain or one of these special values:

- **ACCEPT** – will allow the packet to pass through.
- **DROP** – will not let the packet pass through.
- **RETURN** – stops the packet from traversing through a chain and tell it to go back to the previous chain.

In this iptables tutorial, we are going to work with one of the default tables, called **filter**. It consists of three chains:

- **INPUT** – controls incoming packets to the server.
- **FORWARD** – filters incoming packets that will be forwarded somewhere else.
- **OUTPUT** – filter packets that are going out from your server.

Before we begin this guide, make sure you have SSH **root** or **sudo** access to your machine that runs on Ubuntu 16.04 or up. You can establish the connection through **PuTTY** (Windows) or terminal shell (Linux, macOS). If you own Hostinger VPS, you can get the SSH login details on the Servers tab of hPanel.

iptables rules only apply to **ipv4**. If you want to set up a firewall for the **ipv6** protocol, you will need to use **ip6tables** instead.

How to Install and Use Iptables Linux Firewall

We will divide this iptables tutorial into three steps. First, you will learn how to install the tool on Ubuntu. Secondly, we are going to show you how to define the rules. Lastly, we will guide you to make persistent changes in iptables.

Step 1 — Installing Iptables

Iptables comes pre-installed in most Linux distributions. However, if you don't have it in Ubuntu/Debian system by default, follow the steps below:

1. Connect to your server via SSH. If you don't know, you can read our **SSH tutorial**.
2. Execute the following command one by one:

sudo apt-get update

sudo apt-get install iptables

3. Check the status of your current iptables configuration by running:

sudo iptables -L -v

4. Here, the -L option is used to list all the rules, and -v is for showing the info in a more detailed format. Below is the example output:

Chain INPUT (policy ACCEPT 0 packets, 0 bytes)

pkts bytes target prot opt in out source destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)

pkts bytes target prot opt in out source destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)

pkts bytes target prot opt in out source destination

You will now have the Linux firewall installed. At this point, you can notice that all chains are set to **ACCEPT** and have no rules. This is not secure since any packet can come through without filtering.

Don't worry. We'll tell you how to define rules on the next step of our iptables tutorial.

Step 2 – Defining Chain Rules

Defining a rule means appending it to the chain. To do this, you need to insert the **-A** option (**Append**) right after the iptables command, like so:
sudo iptables -A

It will alert iptables that you are adding new rules to a chain. Then, you can combine the command with other options, such as:

- **-i** (**interface**) — the network interface whose traffic you want to filter, such as eth0, lo, ppp0, etc.
- **-p** (**protocol**) — the network protocol where your filtering process takes place. It can be either **tcp**, **udp**, **udplite**, **icmp**, **sctp**, **icmpv6**, and so on. Alternatively, you can type **all** to choose every protocol.
- **-s** (**source**) — the address from which traffic comes from. You can add a hostname or IP address.
- **–dport** (**destination port**) — the destination port number of a protocol, such as **22** (**SSH**), **443** (**https**), etc.
- **-j** (**target**) — the target name (**ACCEPT**, **DROP**, **RETURN**). You need to insert this every time you make a new rule.

If you want to use all of them, you must write the command in this order:
sudo iptables -A <chain> -i <interface> -p <protocol (tcp/udp) > -s <source> --dport <port no.> -j <target>

Once you understand the basic syntax, you can start configuring the firewall to give more security to your server. For this iptables tutorial, we are going to use the **INPUT** chain as an example.

Enabling Traffic on Localhost

To allow traffic on localhost, type this command:
sudo iptables -A INPUT -i lo -j ACCEPT

For this iptables tutorial, we use **lo** or **loopback** interface. It is utilized for all communications on the localhost. The command above will make sure that the connections between a database and a web application on the same machine are working properly.

Enabling Connections on HTTP, SSH, and SSL Port

Next, we want **http** (port **80**), **https** (port **443**), and **ssh** (port **22**) connections to work as usual. To do this, we need to specify the protocol (**-p**) and the corresponding port (**–dport**). You can execute these commands one by one:

```
sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT

sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT

sudo iptables -A INPUT -p tcp --dport 443 -j ACCEPT
```

It's time to check if the rules have been appended in iptables:

```
sudo iptables -L -v
```

It should return with the results below which means all TCP protocol connections from the specified ports will be accepted:

```
    0     0 ACCEPT     tcp  --  any    any     anywhere             anywhere             tcp dpt:ssh
    0     0 ACCEPT     tcp  --  any    any     anywhere             anywhere             tcp dpt:http
    0     0 ACCEPT     tcp  --  any    any     anywhere             anywhere             tcp dpt:https
```

Filtering Packets Based on Source

Iptables allows you to filter packets based on an IP address or a range of IP addresses. You need to specify it after the **-s** option. For example, to accept packets from **192.168.1.3**, the command would be:

```
sudo iptables -A INPUT -s 192.168.1.3 -j ACCEPT
```

You can also reject packets from a specific IP address by replacing the **ACCEPT** target with **DROP**.

```
sudo iptables -A INPUT -s 192.168.1.3 -j DROP
```

If you want to drop packets from a range of IP addresses, you have to use the **-m** option and **iprange** module. Then, specify the IP address range with **–src-range**. Remember, a hyphen should separate the range of ip addresses without space, like this:

```
sudo iptables -A INPUT -m iprange --src-range 192.168.1.100-192.168.1.200 -j DROP
```

Dropping all Other Traffic

It is crucial to use the **DROP** target for all other traffic after defining **–dport** rules. This will prevent an unauthorized connection from accessing the server via other open ports. To achieve this, simply type:
sudo iptables -A INPUT -j DROP

Now, the connection outside the specified port will be dropped.

Deleting Rules

If you want to remove all rules and start with a clean slate, you can use the **-F** option (**flush**):
sudo iptables -F

This command erases all current rules. However, to delete a specific rule, you must use the -D option. First, you need to see all the available rules by entering the following command:
sudo iptables -L --line-numbers

You will get a list of rules with numbers:
Chain INPUT (policy ACCEPT)

num target prot opt source destination

1 ACCEPT all -- 192.168.0.4 anywhere

2 ACCEPT tcp -- anywhere anywhere tcp dpt:https

3 ACCEPT tcp -- anywhere anywhere tcp dpt:http

4 ACCEPT tcp -- anywhere anywhere tcp dpt:ssh

To delete a rule, insert the corresponding chain and the number from the list. Let's say for this iptables tutorial, we want to get rid of **rule number three** of the **INPUT** chain. The command should be:
sudo iptables -D INPUT 3

Step 3 – Persisting Changes

The iptables rules that we have created are saved in memory. That means we have to redefine them on reboot. To make these changes persistent after restarting the server, you can use this command:
sudo /sbin/iptables-save

It will save the current rules on the system configuration file, which will be used to reconfigure the tables every time the server reboots.

Note that you should always run this command every time you make changes to the rules. For example, if you want to disable iptables, you need to execute these two lines:

sudo iptables -F

sudo /sbin/iptables-save

You will see the following results:

```
root@vps42707999:~# sudo /sbin/iptables-save
# Generated by iptables-save v1.6.1 on Tue Nov 19 11:38:18 2019
*nat
:PREROUTING ACCEPT [645426:37524813]
:INPUT ACCEPT [558190:28836818]
:OUTPUT ACCEPT [263056:18809892]
:POSTROUTING ACCEPT [263056:18809892]
:OUTPUT_direct - [0:0]
:POSTROUTING_ZONES - [0:0]
:POSTROUTING_ZONES_SOURCE - [0:0]
:POSTROUTING_direct - [0:0]
:POST_public - [0:0]
:POST_public_allow - [0:0]
```

Conclusion:

Personal Firewall using iptables was set up.