HW2

1.a)

Wireshark:

-Uses more CPU and consumes more memory compared to dumpcap and tcpdump.

-Powerful sniffer which can decode lots of protocols and has lots of filters.

-It has Graphical User Interface and packet output is interactive and screen oriented.

-Has built in protocol dissectors and many, many protocols which are fully dissected and displayed.

-Gets the packet input directly from n/w interface in libpcap format file and many other formats.

-The amount of packet drops are high compared to dumpcap and tcpdump

TCPdump:

-Less amount of CPU usage compared to wireshark.

-Protocol decoding is limited compared to wireshark.

-Does not have a graphical user interface.

-Gets the packet input directly from n/w interface in libpcap format file.

-The amount of packet drops are very less compared to dumpcap and wireshark.

Dumpcap:

-This is called by Wireshark for capture.

-This uses less CPU than Wiresharka and more CPU than TCPdump.

-Does not have a graphical user interface.

- Gets the packet input directly from n/w interface in libpcap format file.

- Captures packets from a live network and writes the packets to a file. Output is in pcapng (default) or libpcap format

-The amount of packet drops are less compared to wireshark and more compared to TCPdump.

1.b.)

How the above tools work?

-Basically the capturing module is placed between the NIC driver and higher network layer protocols in the kernel.

-The capturing framework filters the packets before the come up the network stack.

-The framework compiles the high level filter specification into low level code that filters the packets at driver level.

-This kernel module is called Berkeley packet Filter(BPF)
-The commands from the CML interface are given to the BPF which converts the high level filter specification into low level code that filters packet at driver.
-Packets to and from the capturing machine are passed through the BPF.
-These packets are then given to the tool, which are then decoded, displayed and dumped to disk.
A compiler to convert higher level model to machine level model.

--Is it possible to capture both incoming and outgoing packets with this module?
-In an operating system the NIC driver is only responsible for direct hardware access (receiving and transmitting of network frames, generating interrupts etc.). If the OS wants to send a frame, it (actually a higher level driver) will create the ethernet frame and pass that to the hardware NIC driver in order to send it. Due to this architecture it is possible to intercept outgoing frames before they are sent, as the capture driver is placed 'somewhere' between the NIC driver (hardware) and the upper layer drivers (ethernet, ip, tcp, etc.).
-Systems with BPF (*BSD, OS X): The capture mechanism (BPF) is called by the driver. So it is possible to both incoming and outgoing packets with this module.

BPF language:
-The BPF filter language starts from a basic predicate, which is true if the specified packet field equals the indicated value.
Predicate: field val
Field: protocol die selector
Protocol – Either ip/tcp etc..
Dir – either src/dst
Selector – host/network/port
-A BPF language and parser to translate it and gen_cmp() generates code to compare a packet field to a value.
-The above tools use packet capture library called libpcap and operate on interchangeable file format for packet capture called pcap.
Libpcap:
-The compiler system and the filtering engine is pulled out of tcpdump to create API and reusable library to build packet capture apps like Wireshark.

-This is released as libpcap. Libpcap is an architecture and optimization methodology for packet capture.

Pcap:

The Packet Capture library provides a high level interface to packet capture systems. All packets on the network, even those destined for other hosts, are accessible through this mechanism. It also supports saving captured packets to a ``savefile'', and reading packets from a ``savefile''.

2.)

The storage required for Wireshark capture is the highest compared to storage required by the TCPdump and Dumpcap. This is because, by default the Wireshark captures the entire content of the packet as it is received across the wire. The average Packet Size in my capture was 539 bytes.

The storage required by TCPdump capture is highest compared to Dumpcap. The Avg packet size of TCPdump capture is 369 bytes while the Avg packet size of the Dumpcap capture is 270 bytes. This is because dumpcap just captures the packet and does not do any protocol decoding while the TCPdump does protocol decoding.

TCPdump takes less time for the capture. This is because, compared to wireshark, there is no overhead of extra processing per packet in TCPdump. The amount of packet drops by kernel are also less compared to wireshark.

3.)

Wireless: All the traffic is captured at the second floor of the library between 7PM and 8PM.

-HTTP traffic

The percentage of HTTP traffic shown by Wireshark, TCPdump, and Dumpcap is 0.034%, 0.0742% and 0.0264% respectively. I think it is this low because of the HTTP data is being cached locally on the users machines.

-Tcp traffic

The percentage of TCP traffic shown by Wireshark, TCPdump, and Dumpcap is 9.5%, 14.7% and 5.62% respectively. This is the second highest amount of traffic out of all the traffic types.

-Wifi(802.11) traffic

The percentage of Wifi traffic shown by Wireshark, TCPdump, and Dumpcap is 86.5%, 81.3% and 91.2% respectively. This is the highest traffic type out of all the types. As the capture is on wireless interface and as I'm listening in monitor mode, the tool was capturing all the 802.11 packets to all the machines in the network from the accesspoint.

-The amount of retransmissions
The percentage of retransmitted packets shown by Wireshark, TCPdump, and Dumpcap out of all the traffic is 2.9%, 4.09%, and 2.22% respectively.

-UDP traffic
The percentage of UDP traffic shown by Wireshark, TCPdump, and Dumpcap out of all the traffic is 1.738%, 0.1522%, and 0.2472% respectively.

Wired:
The percentage of retransmitted packets in this case is very less compared to wireless interface. This is because there is low packet loss in case wired interface.
-The percentage of retransmitted packets
The percentage of retransmitted packets with wired interface as shown by Wireshark, TCPdump, and Dumpcap out of all the traffic is 0.13%, 0.038%, and 0.166% respectively.

-HTTP traffic
The percentage of HTTP traffic shown by Wireshark, TCPdump, and Dumpcap is 0.019%, 0.1% and 0.34% respectively. I think it is this low because of the HTTP data is being cached locally on the users machines.

-Tcp traffic
The percentage of TCP traffic shown by Wireshark, TCPdump, and Dumpcap is 11.34%, 38.786% and 17.946% respectively.

-UDP traffic
The percentage of UDP traffic shown by Wireshark, TCPdump, and Dumpcap out of all the traffic is 8.013%, 2.97%, and 4.806% respectively.

4.)

i.)ICMP ping reachability:

Google.com: 2 packets sent and 2 packets received. 100% reachability and Latency is 8ms.
Fb.com: 2 packets sent and 2 packets received. 100% eachability and Latency is 4.2ms
Host in my same network(172.25.47.200): 2 packets received. 100% reachability and Latency is 80ms.

ii.)Nmap port scan:
-Performed this with target as my friend's PC in the same network.
```
nmap -Pn  172.25.47.200
```
The result is as follows:
```
Nmap scan report for 172.25.47.200
Host is up (0.034s latency).
Not shown: 995 filtered ports
PORT      STATE SERVICE
135/tcp   open  msrpc
139/tcp   open  netbios-ssn
445/tcp   open  microsoft-ds
1720/tcp  open  H.323/Q.931
49155/tcp open  unknown
Nmap done:1 IP address(1 host up)scanned in 29.83s
```

-Performed this with target as my friend's PC in India.
```
nmap -Pn 192.168.1.106
```
The result is as follows:
```
Nmap scan report for 192.168.1.106
Host is up (0.065s latency).
Not shown: 998 filtered ports
PORT     STATE  SERVICE
113/tcp  closed ident
1720/tcp open   H.323/Q.931
Nmap done: 1 IP address (1 host up) scanned in 29.64
seconds
```

iii.)Nmap TCP/IP fingerprinting
with target as my friends computer: 172.25.47.200
sudo nmap -O -v 172.25.47.200

The results are as follows:
```
MAC Address: 00:90:0B:2C:FC:FB (Lanner Electronics)

Warning: OSScan results may be unreliable because we
could not find at least 1 open and 1 closed port
```

```
Device type: general purpose

Running: Linux 2.6.X
OS CPE: cpe:/o:linux:linux_kernel:2.6.34
OS details: Linux 2.6.34
Uptime guess: 41.436 days (since Thu Mar  5 05:09:40
2015)
Network Distance: 1 hop
TCP Sequence Prediction: Difficulty=198 (Good luck!)
IP ID Sequence Generation: All zeros
```

The above results did not show the correct OS on the target machine. However, it gave a warning about this.

-With target as my friends computer in India
sudo nmap -Pn -O -v 192.168.1.106

The results are as follows:

```
Device type: switch|firewall|WAP|general
purpose|broadband router|specialized
Running: Avaya embedded, D-Link embedded, Fortinet
embedded, Linksys embedded, Microsoft Windows 3.X,
Motorola VxWorks 5.X, Siemens embedded
OS CPE: cpe:/h:avaya:p580 cpe:/h:dlink:des-3010f
cpe:/h:dlink:dgs-3010g cpe:/h:linksys:wga54g
cpe:/o:microsoft:windows cpe:/o:motorola:vxworks:5
cpe:/h:siemens:simatic_tdc
Too many fingerprints match this host to give
specific OS details
```

Unlike before, when I was fingerprinting my friend's PC in the same network as mine, In this case I was trying a remote PC in different country. So I think the fingerprinting was done for a local switch through which the pings have passed

iv.)Reverse DNS:
The reverse DNS was performed on the IP address of UH.

```
dig -x 129.7.97.54
```

The query returned the web addresses of a bunch of websites which are part of the UH web site "www.uh.edu"
The result is as follows:

```
;; QUESTION SECTION:
```

```
;54.97.7.129.in-addr.arpa.  IN  PTR

;; ANSWER SECTION:
54.97.7.129.in-addr.arpa. 262  IN  PTR
    uscholars.uh.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR
    www.uhsystem.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR uhsystem.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR
    minimester.uh.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR career.uh.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR housing.uh.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR txccrn.uh.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR gapps-
dev.uh.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR
    keephoustonred.com.
.
.
.
.
54.97.7.129.in-addr.arpa. 262  IN  PTR
    digitalmedia.tech.uh.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR
    campusnet.uh.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR hrm.uh.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR
    pharmacy.uh.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR
    cincoranch.uh.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR
    sugarland.uh.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR
    www.uhsa.uh.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR uh.edu.
54.97.7.129.in-addr.arpa. 262  IN  PTR
    coogsgetconsent.org.
54.97.7.129.in-addr.arpa. 262  IN  PTR sa.uh.edu.
```

V.) Traceroute:
Cmd: traceroute www.google.com

Using the tracert command as shown above, we're asking traceroute to show us the path from the local computer all the way to the network device with the hostname*www.google.com*.

```
traceroute to www.google.com (64.233.169.147), 64
hops max, 52 byte packets
 1  g3-cs2.sw-e.uh.edu (172.16.255.32)  1.010 ms
0.808 ms  0.767 ms
 2  172.25.255.253 (172.25.255.253)  43.231 ms
10.272 ms  5.421 ms
 3  flavia-t3-9.p2p.e.uh.edu (172.16.255.241)  1.312
ms  1.218 ms  1.068 ms
 4  172.16.96.91 (172.16.96.91)  1.221 ms  1.731 ms
9.495 ms
 5  caesar-vlan1915.uh.edu (129.7.0.253)  1.374 ms
1.268 ms  1.325 ms
 6  hardy-int-rtr1.setg.net (198.32.231.153)  1.586
ms  4.449 ms  2.149 ms
.
.
.
.

    66.249.95.198 (66.249.95.198)  15.705 ms
13  72.14.237.123 (72.14.237.123)  19.410 ms
    209.85.249.69 (209.85.249.69)  18.601 ms
    66.249.95.218 (66.249.95.218)  43.966 ms
14  209.85.244.190 (209.85.244.190)  17.719 ms
    72.14.239.156 (72.14.239.156)  19.705 ms
    209.85.244.190 (209.85.244.190)  15.376 ms
15  * * *
16  64.233.169.147 (64.233.169.147)  17.712 ms
15.138 ms  13.972 ms
```

In this example we can see that traceroute identified fifteen network devices including our router at **172.16.255.32** and all the way through to the *target* of *www.google.com* which we now know uses the public IP address of **64.233.169.147**