Lab 1a: Packet Capture (Network Interface)

Details

Aim: To provide a foundation in reading data packets

Activities

If Visual Studio is installed on your machine, download the following solution [1]:

↳http://www.dcs.napier.ac.uk/~bill/WinPCap1.zip

It has the following code [1]:

```
using System;
using Tamir.IPLib;

namespace NapierCapture
{

   public class ShowDevices
   {

      public static void Main(string[] args)
      {
         string verWinPCap =null;
         int count=0;

         verWinPCap= Tamir.IPLib.Version.GetVersionString();

         PcapDeviceList getNetConnections = SharpPcap.GetAllDevices();

         Console.WriteLine("WinPCap Version: {0}", verWinPCap);

         Console.WriteLine("Connected devices:\r\n");

         foreach(PcapDevice net in getNetConnections)
         {
            Console.WriteLine("{0}) {1}",count,net.PcapDescription);
            Console.WriteLine("\tName:\t{0}",net.PcapName);
            Console.WriteLine("\tMode:\t\t\t{0}",net.PcapMode);
            Console.WriteLine("\tIP Address: \t\t{0}",net.PcapIpAddress);
            Console.WriteLine("\tLoopback: \t\t{0}",net.PcapLoopback);

            Console.WriteLine();
            count++;
         }

         Console.Write("Press any <RETURN> to exit");
         Console.Read();

      }
   }
}
```

Run the program, and verify that it produces a list of the available network cards, such as:

```
WinPCap Version: 1.0.2.0
Connected devices:
```

```
0) Realtek RTL8169/8110 Family Gigabit Ethernet NIC
    (Microsoft's Packet Scheduler)
        Name:   \Device\NPF_{A22E93C1-A78D-4AFE-AD2B-517889CE42D7}
        Mode:                   Capture
        IP Address:             192.168.2.1
        Loopback:               False

1) Intel(R) PRO/Wireless 2200BG Network Connection (Microsoft's Packet Scheduler)
        Name:   \Device\NPF_{044B069D-B90A-4597-B99E-A68C422D5FE3}
        Mode:                   Capture
        IP Address:             192.168.1.101
        Loopback:               False
```

List the network cards in your machine:

Next update the code so that it displays the information on the network connections [1]:

```
foreach(PcapDevice net in getNetConnections)
{
  Console.WriteLine("{0}) {1}",count,net.PcapDescription);

  NetworkDevice netConn = (NetworkDevice)net;

  Console.WriteLine("\tIP Address:\t\t{0}",netConn.IpAddress);
  Console.WriteLine("\tSubnet Mask:\t\t{0}",netConn.SubnetMask);
  Console.WriteLine("\tMAC Address:\t\t{0}",netConn.MacAddress);
  Console.WriteLine("\tDefault Gateway:\t{0}",netConn.DefaultGateway);
  Console.WriteLine("\tPrimary WINS:\t\t{0}",netConn.WinsServerPrimary);
  Console.WriteLine("\tSecondary WINS:\t\t{0}",netConn.WinsServerSecondary);
  Console.WriteLine("\tDHCP Enabled:\t\t{0}",netConn.DhcpEnabled);
  Console.WriteLine("\tDHCP Server:\t\t{0}",netConn.DhcpServer);
  Console.WriteLine("\tDHCP Lease Obtained:\t{0}",netConn.DhcpLeaseObtained);
  Console.WriteLine("\tDHCP Lease Expires:\t{0}",netConn.DhcpLeaseExpires);
  Console.WriteLine();
  count++;
}
```

A sample run shows the details of the network connections [1]:

```
1) Intel(R) PRO/Wireless 2200BG Network Connection (Microsoft's Packet Scheduler)
        IP Address:             192.168.1.101
        Subnet Mask:            255.255.255.0
        MAC Address:            0015003402F0
        Default Gateway:        192.168.1.1
        Primary WINS:           0.0.0.0
        Secondary WINS:         0.0.0.0
        DHCP Enabled:           True
        DHCP Server:            192.168.1.1
        DHCP Lease Obtained:    03/01/2006 10:44:40
        DHCP Lease Expires:     04/01/2006 10:44:40
```

List the details of the connections on your PC:

[1] This code is based on the code wrapper for WinPCap developed by T.Gal [http://www.thecodeproject.com/csharp/sharppcap.asp].

Lab 1b: Packet Capture (Filtering)

Details

Aim: To provide an understanding of events in reading data packets

Activities

Using the previous solution from Lab 1, update with the following code [1]. In this case the 2nd connection is used (getNetConnections[1]) in a promiscuous mode (change, as required, depending on your network connection). USE THE CONNECTION WHICH IS THE ETHERNET CONNECTION.

↳http://www.dcs.napier.ac.uk/~bill/WinPCap2.zip

```
using System;
using Tamir.IPLib;
using Tamir.IPLib.Packets;

namespace NapierCapture
{
   public class CapturePackets
   {
      public static void Main(string[] args)
      {
         PcapDeviceList getNetConnections = SharpPcap.GetAllDevices();

         // network connection 1 (change as required)
         NetworkDevice netConn = (NetworkDevice)getNetConnections[1];
         PcapDevice device = netConn;

         // Define packet handler
         device.PcapOnPacketArrival +=
            new SharpPcap.PacketArrivalEvent(device_PcapOnPacketArrival);

         //Open the device for capturing
         //true -- means promiscuous mode
         //1000 -- means a read wait of 1000ms
         device.PcapOpen(true, 1000);

         Console.WriteLine("Network connection: {0}",device.PcapDescription);

         //Start the capturing process
         device.PcapStartCapture();

         Console.Write("Press any <RETURN> to exit");
         Console.Read();

         device.PcapStopCapture();
         device.PcapClose();
      }
      private static void device_PcapOnPacketArrival(object sender, Packet packet)
      {
         DateTime time = packet.PcapHeader.Date;
         int len = packet.PcapHeader.PacketLength;
         Console.WriteLine("{0}:{1}:{2},{3} Len={4}",time.Hour, time.Minute,
                     time.Second, time.Millisecond, len);
      }
   }
}
```

Run the program, and produce some network traffic and versify that it is capturing packets, such as:

```
13:17:56,990 Len=695
13:17:57,66 Len=288
13:17:57,68 Len=694
13:18:4,363 Len=319
13:18:4,364 Len=373
13:18:4,364 Len=371
13:18:4,365 Len=375
13:18:4,366 Len=367
```

Did it capture packets? **Yes/No**

Update the code with a filter. In the following case an **IP** and **TCP** filter is used [1]:

```
device.PcapOpen(true, 1000);

Console.WriteLine("Network connection: {0}",device.PcapDescription);

string filter = "ip and tcp";

//Associate the filter with this capture
device.PcapSetFilter( filter );

//Start the capturing process
device.PcapStartCapture();
```

Generate some data traffic, such as loading a Web page, and show that the program is capturing the data packets.

Did it capture packets? **Yes/No**

Next update the filter so that it only captures **ICMP** packets, such as:

```
string filter = "icmp";
```

Generate some data traffic, and prove that it does not capture the packets. Now ping a node on your network, such as:

```
Ping 192.168.1.102
```

And prove that it captures the data packets, such as:

```
13:40:47,761 Len=74
13:40:48,756 Len=74
13:40:48,759 Len=74
13:40:49,757 Len=74
13:40:49,760 Len=74
13:40:50,757 Len=74
```

Did it capture ICMP packets? **Yes/No**

[1] This code is based on the code wrapper for WinPCap developed by T.Gal [http://www.thecodeproject.com/csharp/sharppcap.asp].

Lab 1c: Packet Capture (IDS)

Details

Aim: To provide define the usage of an intrusion detection system

Activities

1. The WinPcap library can be used to read the source and destination IP addresses and TCP ports. For this the **TCPPacket** class is used. Initially modify the program in Lab 2 so that it now displays the source and destination IP and TCP ports [1]:

↳http://www.dcs.napier.ac.uk/~bill/WinPCap3.zip

```
private static void device_PcapOnPacketArrival(object sender, Packet packet)
{
   if(packet is TCPPacket)
   {
      DateTime time = packet.PcapHeader.Date;
      int len = packet.PcapHeader.PacketLength;

      TCPPacket tcp = (TCPPacket)packet;
      string srcIp = tcp.SourceAddress;
      string dstIp = tcp.DestinationAddress;
      int srcPort = tcp.SourcePort;
      int dstPort = tcp.DestinationPort;

      Console.WriteLine("{0}:{1} -> {2}:{3}", srcIp, srcPort, dstIp, dstPort);
   }
}
```

A sample run, using a Web browser connected to google.com gives:

```
84.53.143.151:80 -> 192.168.1.101:3582
84.53.143.151:80 -> 192.168.1.101:3582
192.168.1.101:3582 -> 84.53.143.151:80
```

Where it can be seen that the WWW server TCP port is 80, and the local port is 3582. Run the program, and generate some network activity, and determine the output.

Determine the output of the test run:

2. Modify the program in 3.12.1, so that it only displays traffic which is *distended* for a Web server. Prove its operation.

How was the code modified:

3. Next modify the code so that it detects only ICMP packets (using the **ICMPPacket** class), and displays the source and the destination addresses, along with the TTL (time-to-live) value [1]:

```
private static void device_PcapOnPacketArrival(object sender, Packet packet)
{
   if(packet is ICMPPacket)
   {
      DateTime time = packet.PcapHeader.Date;
      int len = packet.PcapHeader.PacketLength;

      ICMPPacket icmp = (ICMPPacket)packet;
      string srcIp=icmp.DestinationAddress;
      string dstIp=icmp.SourceAddress;
      string ttl=icmp.TimeToLive.ToString();

      Console.WriteLine("{0}->{1} TTL:{2}", srcIp, dstIp, ttl);
   }
}
```

A sample run is shown next for a ping on node 192.168.1.102:

```
Press any <RETURN> to exit
192.168.1.101->192.168.1.102 TTL:128
192.168.1.102->192.168.1.101 TTL:128
192.168.1.101->192.168.1.102 TTL:128
```

Run the program, and ping a node on the network. What is the output, and why does it show three responses for every ping:

4. Modify the program in 3.12.2, so that it displays the Ethernet details of the data frame, such as [4]:

```
private static void device_PcapOnPacketArrival(object sender, Packet packet)
{
   if( packet is EthernetPacket )
   {
      EthernetPacket etherFrame = (EthernetPacket)packet;
      Console.WriteLine("At: {0}:{1}: MAC:{2} -> MAC:{3}",
         etherFrame.PcapHeader.Date.ToString(),
         etherFrame.PcapHeader.Date.Millisecond,
         etherFrame.SourceHwAddress,
         etherFrame.DestinationHwAddress);
   }
}
```

5. It is possible to read the contents of the data package by converting it to a byte array (using the Data property), and then convert it to a string, such as:

```
private static void device_PcapOnPacketArrival(object sender, Packet packet)
{
   if(packet is TCPPacket)
```

```
    {
       DateTime time = packet.PcapHeader.Date;
       int len = packet.PcapHeader.PacketLength;

       TCPPacket tcp = (TCPPacket)packet;

       byte [] b = tcp.Data;

       System.Text.ASCIIEncoding format = new System.Text.ASCIIEncoding();

       string s = format.GetString(b);

       s=s.ToLower();

       if (s.IndexOf("intel")>0) Console.WriteLine("Intel found...");
    }
}
```

The above code detects the presence of the word Intel in the data packet. Run the program, and then load a site with the word Intel in it, and prove that it works, such as for:


```
Intel found...
Intel found...
```

Did the code work:

6. It is then possible to filter for source and destination ports, and with source and destination addresses. For example, the following detects the word Intel on the destination port of 80:

```
private static void device_PcapOnPacketArrival(object sender, Packet packet)
{
   if (packet is TCPPacket)
   {
      DateTime time = packet.PcapHeader.Date;
      int len = packet.PcapHeader.PacketLength;

      TCPPacket tcp = (TCPPacket)packet;
      int destPort = tcp.SourcePort;

      byte [] b = tcp.Data;

      System.Text.ASCIIEncoding format = new System.Text.ASCIIEncoding();

      string s = format.GetString(b);

      s=s.ToLower();

      if (destPort==80 && (s.IndexOf("intel")>0))
         Console.WriteLine("Intel found in outgoing on port 80...");
   }
}
```

Did the code work:

7. A key indication of network traffic is in the TCP flags. The following determines when the SYN flag is detected, and also the SYN, ACK flags:

```
if(packet is TCPPacket)
{
   DateTime time = packet.PcapHeader.Date;
   int len = packet.PcapHeader.PacketLength;

   TCPPacket tcp = (TCPPacket)packet;
   int destPort = tcp.SourcePort;

   if (tcp.Syn) Console.WriteLine("SYN request");
   if (tcp.Syn && tcp.Ack) Console.WriteLine("SYN and ACK");

}
```

Prove the operation of the code, and modify it so that it detects a SYN request to a Web server (port: 80), and displays the destination IP address of the Web server.

Outline the code used:

8. Modify the code in 7 so that it displays all the flags for data packets.

[1] This code is based on the code wrapper for WinPCap developed by T.Gal [http://www.thecodeproject.com/csharp/sharppcap.asp].

Lab 1d: Packet Capture (IDS) – ARP Detection

Details

Aim: To provide define the capture of ARP information

Activities

1. The ARP protocol is important on networks, as it allows a node to determine the MAC address of a destination node on the same network. For security it is important, as it gives information on the activity on the local network. In this lab ARP packets will be captured, and then displayed for their basic information. The solution can be found at:

↳http://www.dcs.napier.ac.uk/~bill/WinPCap4.zip

2. The basic format of the ARP header is:

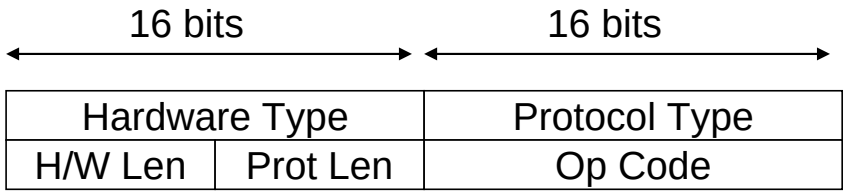

Figure 1: ARP header

Thus a program to capture the ARP packets is given next. Notice that the byte array is read for the first two bytes for the hardware type, and the next two for the protocol type [1]:

```
using System;
using Tamir.IPLib;
using Tamir.IPLib.Packets;

namespace NapierCapture
{
   public class CapturePackets
   {
      public static void Main(string[] args)
      {
         PcapDeviceList getNetConnections = SharpPcap.GetAllDevices();

         // network connection 1 (change as required)
         NetworkDevice netConn = (NetworkDevice)getNetConnections[1];
         PcapDevice device = netConn;

         // Define packet handler
         device.PcapOnPacketArrival +=
            new SharpPcap.PacketArrivalEvent(device_PcapOnPacketArrival);

         device.PcapOpen(true, 1000);
         Console.WriteLine("Network connection: {0}",device.PcapDescription);

         //Start the capturing process
```

```
            device.PcapStartCapture();

            Console.WriteLine("Press any <RETURN> to exit");
            Console.Read();

            device.PcapStopCapture();
            device.PcapClose();
        }
        private static void device_PcapOnPacketArrival(object sender, Packet packet)
        {

            if(packet is ARPPacket)
            {
                  byte [] b = packet.Header;

                  int type = b[1] + (b[0]<<8);

                  int protocol = b[3] + (b[2]<<8);

                  int opcode =  b[7] + (b[6]<<8);

               Console.WriteLine("ARP: Hardware type {0}, protocol {1}, op-code: {2}",
                           type,protocol,opcode);
        }
    }
}
```

Run the code, and ping a node on your network (one which you have not previously accessed for a while, or not at all), and examine the output:

Output of the program:

Did it detect the ARP packets:

What where the ARP types (from the op-code [2][1]):

3. Modify the code so that it displays the other fields in the ARP header.

4. Modify the code so that it displays the actual ARP type, rather than the code, Such as with:

[1] Note: For Ethernet, the **type** is normal set to 1 [2]. The **protocol** type for IP is 0x8000 (2048), and the table for the op-code is:

1 Request
2 Reply
3 Request Reverse
4 Rely Request

```
Console.Write("ARP: Hardware type {0}, protocol {1}, ",type,protocol);
if (opcode==1) Console.Write("{0}",opcode);
else if (opcode==2) …
```

References

[1] This code is based on the code wrapper for WinPCap developed by T.Gal [http://www.thecodeproject.com/csharp/sharppcap.asp].

[2] http://www.networksorcery.com/enp/protocol/arp.htm