Packet Sniffing and Decoding

Gowtham Munukutla
CED15I019

Goal

To capture live with an open wireless interface and decode the various layers in each packet displaying all its headers.

Implementation

Using a packet processing library, go-packet and its submodules in Golang.

Documentation

Required Variables

```
var (
    device      string = "wlp2s0"
    snapshotLen int32  = 1024
    promiscuous bool   = false
    err         error
    timeout     time.Duration = 30 * time.Second
    handle      *pcap.Handle
)
```

`device` is the interface which is going to be opened for live capture.
`snapshotLen` is the maximum size to read for each packet.
`Handle` is like a file pointer but to the interface.

Using these variables we open the interface for live capture of the packet with the following snippet.

>>>>

```
handle, err = pcap.OpenLive(device, snapshotLen, promiscuous, timeout)
```

>>>>

Package `pcap` allows users of gopacket to read packets off of the wire or from pcap files. The above line returns an `err` in case the operation wasn't successful.

Errors can occur in the following scenarios :

1. Insufficient permissions to open the specified network interface.
2. There is no such specified interface.
   For ex: `wlan and wlp2s0, eth0 and enp3s0`

We log the error in that case.

>>>>

```
if err != nil {
    log.Fatal(err)
}
```

>>>>

Now we create a new packet source from the handle we acquired.

>>>>

```
packetSource := gopacket.NewPacketSource(handle, handle.LinkType())
```

>>>>

`LinkType` is an enumeration of Link Types and acts as a decoder for any link it supports.

`packetSource` reads in packets from a packet source, decodes and returns them.

There are currently two different ways with them we can access to the stream of packets from `packetSource.`

`packetSource.NextPacket()` and `packetSource.Packets()`

Here we're using the second way because it returns a **channel** (analogous to a buffer in layman's terms) which allows easy iteration of packets.

Finally in the following snippet we iterate the channel and print the information of the packet.

```
>>>>
            for packet := range packetSource.Packets() {
                count++
                printPacketInfo(packet, count)
            }
>>>>
```

```
printPacketInfo(packet, count) {}
```

The above functions does all the required decoding and extraction of the headers and prints it in a human readable way which is otherwise returned as a byte stream.

There is a custom package helper library created in the root directory itself for parsing the headers of each layers separately.

```
>>>>
        ethernetLayer := packet.Layer(layers.LayerTypeEthernet)
>>>>
```

For example, the above line of code returns the ethernet layer data from the packet upon specifying the layer we need, by `layers.LayerTypeEthernet.`

In the same way data of other layers are extracted by specifying the respective parameters.

>>>>

```
        tcpLayer := packet.Layer(layers.LayerTypeTCP)

        ipLayer := packet.Layer(layers.LayerTypeIPv4)
```

>>>>

Ethernet Layer

After getting the required layer, we get each packet via iterating the above mentioned channel. Ethernet header has four fields.

>>>>

```
        if ethernetLayer != nil {
            ethernetPacket, _ := ethernetLayer.(*layers.Ethernet)

        // do something with the packet
            ...  ... ...
        }
```

>>>>

IP Layer

>>>>

```
        if ipLayer != nil {
            ipPacket, _ := ipLayer.(*layers.IPv4)

            // do something with the packet
            ...  ... ...
        }
```

>>>>

TCP Layer

```
>>>>
          if tcpLayer != nil {
             tcpPacket, _ := tcpLayer.(*layers.TCP)

             // do something with the packet
             ...   ... ...
          }
>>>>
```

TCP Flags

TCP has a number of boolean flag options in its header. There the library returns an array of flags which is converted into a map with `boolean` values and then iterated to access each of them.

```
>>>>
     func PPTCPFlags(m map[string]bool) {

          for k, v := range m {
               if v {
               // print key(k) if value(v) is true in the header.
               }
          }
     }
>>>>
```

The packet information is written onto the console using a colors library for distinguishing between the headings and the values.

Demo Output

```
HLen: 8 Flags: ACK,PSH, WinSize : 227
Checksum: 56991       URG PTR: 0
        Options: 3
------------------------------------------------------------


------------------------------------------------------------
PACKET  197

Ethernet Layer Found!
DST: 0c:d2:b5:3e:49:24     |    SRC: 2c:6e:85:37:1f:c2     |  Type: IPv4

IP Layer Found!
IP Version: 4   IHL: 5  TOS: 0  Length: 52
     ID: 59107  Flags: DF    OFF: 0
TTL: 64        Pro: TCP         CHS: 50637
           Src: 192.168.1.3
           Dst: 52.40.152.63

TCP Layer Found!
SrcPort: 50346  DstPort: 443(https)
        SEQ: 3843927047
        ACK: 807121246
HLen: 8 Flags: ACK, WinSize : 445
Checksum: 29124       URG PTR: 0
        Options: 3
------------------------------------------------------------


------------------------------------------------------------
PACKET  198

Ethernet Layer Found!
DST: 2c:6e:85:37:1f:c2     |    SRC: 0c:d2:b5:3e:49:24     |  Type: IPv4
```

```
        ACK: 2433562816
HLen: 8 Flags: ACK, WinSize : 392
Checksum: 59655       URG PTR: 0
        Options: 3
------------------------------------------------------------


------------------------------------------------------------
PACKET  71

Ethernet Layer Found!
DST: 2c:6e:85:37:1f:c2     |    SRC: 0c:d2:b5:3e:49:24     |  Type: IPv4

IP Layer Found!
IP Version: 4   IHL: 5  TOS: 0  Length: 52
     ID: 8891  Flags: DF    OFF: 0
TTL: 52        Pro: TCP         CHS: 42177
           Src: 192.30.253.125
           Dst: 192.168.1.3

TCP Layer Found!
SrcPort: 443(https)     DstPort: 37502
        SEQ: 4129626069
        ACK: 4036899806
HLen: 8 Flags: ACK, WinSize : 33
Checksum: 11117       URG PTR: 0
        Options: 3
------------------------------------------------------------


------------------------------------------------------------
PACKET  72

Ethernet Layer Found!
DST: 01:00:5e:00:00:01     |    SRC: 0c:d2:b5:3e:49:24     |  Type: IPv4
```

```
           Dst: 192.30.253.124

TCP Layer Found!
SrcPort: 32790  DstPort: 443(https)
        SEQ: 424824721
        ACK: 1248902907
HLen: 8 Flags: PSH,ACK, WinSize : 302
Checksum: 11298       URG PTR: 0
        Options: 3
-----------------------------------------------------------------


-----------------------------------------------------------------
PACKET  198

Ethernet Layer Found!
DST: 2c:6e:85:37:1f:c2     |    SRC: 0c:d2:b5:3e:49:24     |  Type: IPv4

IP Layer Found!
IP Version: 4   IHL: 5  TOS: 0  Length: 52
     ID: 65075  Flags: DF    OFF: 0
TTL: 52        Pro: TCP         CHS: 51529
           Src: 192.30.253.124
           Dst: 192.168.1.3

TCP Layer Found!
SrcPort: 443(https)     DstPort: 32790
        SEQ: 1248902907
        ACK: 424824756
HLen: 8 Flags: ACK, WinSize : 31
Checksum: 53637       URG PTR: 0
        Options: 3
-----------------------------------------------------------------
```