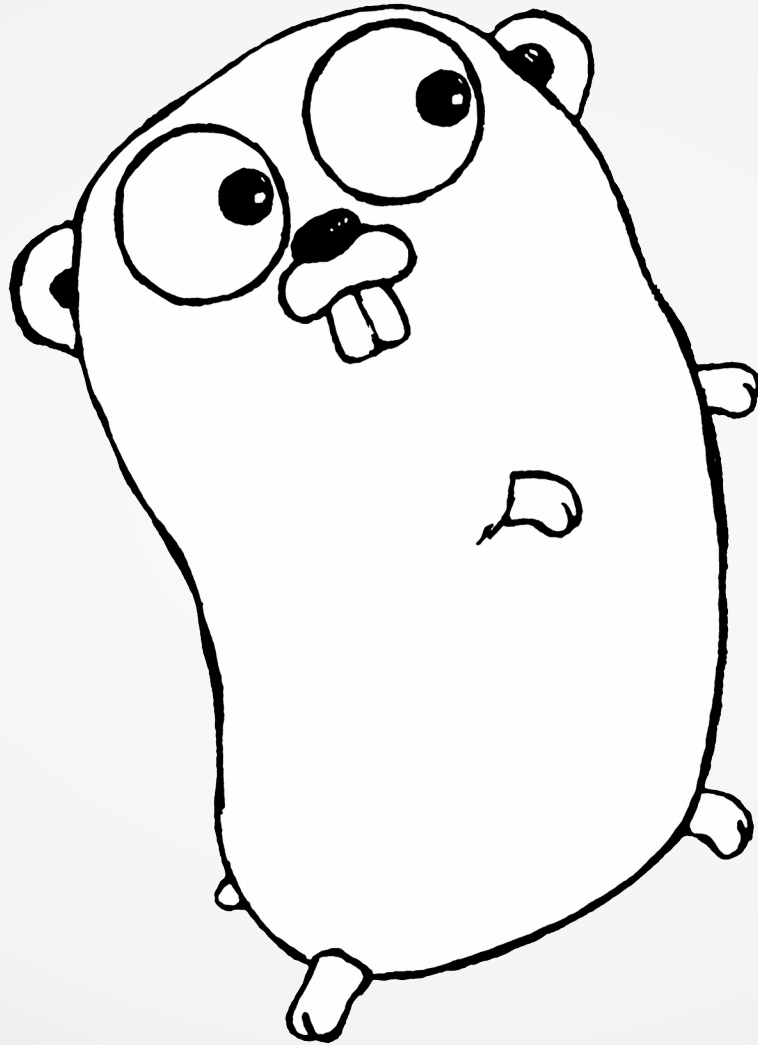


Packet Capturing with Go



About Me

- John Leon, “NanoDano”
- <http://www.devdungeon.com>
- <https://www.github.com/NanoDano>
- @thedevdungeon

What is Packet Capturing

- Wired vs wireless
- Promiscuous mode
- Hubs vs Switches
- Does not block or man-in-the-middle traffic

How Can it be Used?

- App development: app testing, validating encryption
- Reverse engineering an API
- Seeing what traffic goes on in background
- Stealing credentials
- Network administration
- Looking for malicious traffic on network
- Forensics for crime investigations
- Defcon Wall of Sheep

My Motivations

- Hacker by nature
- Want to see what happens under the hood
- Validating authentication mechanisms are encrypted
- Make sure no malicious traffic on my servers
- Understanding how secure traffic is on open wi-fi
- Can my credentials be stolen?
 - Facebook did not use SSL for a long time
 - Neither did OKCupid
 - <https://httpshaming.tumblr.com>

Overview

- Getting a list of network devices
- Capturing packets from a network device
- Saving packets to a file
- Reading packets from a file
- Analyzing packet layers
- Creating custom layers
- Using Berkeley Packet Filters
- Injecting packets
- Following streams

Common Tools

- Wireshark/tshark
- Tcpdump
- Driftnet
- Firesheep

Prerequisites

- libpcap or WinPcap
- Go

libpcap

- C library
- <http://www.tcpdump.org/>
- <http://www.devdungeon.com/content/using-libpcap-c>



WinPcap

- Windows compatible version of libpcap
- <https://www.winpcap.org/>

WinPcap

The industry-standard windows packet capture library



Download
Get WinPcap



Documentation



Support

gopacket

- Utilizes libpcap, but also supports pfring and afpacket
- <https://github.com/google/gopacket>
- <http://www.devdungeon.com/content/packet-capture-injection-and-analysis-gopacket>

 [google](#) / [gopacket](#)

 Code

 Issues 31

 Pull requests 8

 Pulse

 Graphs

Provides packet processing capabilities for Go



Watch ▾

60



Star

650



Fork

127

gopacket Sub-packages

- github.com/google/gopacket
- github.com/google/gopacket/pcap
- github.com/google/gopacket/layers
- github.com/google/gopacket/pcapgo

gopacket Overview

- <https://godoc.org/github.com/google/gopacket>
- Notable types
 - Decoder
 - Flow
 - Layer
 - Packet
 - PacketSource
 - Payload

Get pcap Version

```
import (  
    "fmt"  
    "github.com/google/gopacket/pcap"  
)  
  
func main() {  
    version := pcap.Version()  
    fmt.Println(version)  
}
```

Find Network Device

```
var devices []pcap.Interface  
devices, _ := pcap.FindAllDevs()
```

Network Adapter Struct

```
type Interface struct {  
    Name          string  
    Description   string  
    Addresses     []InterfaceAddress  
}
```


Address Struct

```
type InterfaceAddress struct {  
    IP          net.IP  
    Netmask     net.IPMask  
}
```

Opening Live Device

```
handle, _ := pcap.OpenLive(  
    "eth0",          // device  
    int32(65535),    // snapshot length  
    false,           // promiscuous mode?  
    time.Duration = -1*time.Second, //timeout  
)  
defer handle.Close()
```

Opening pcap File

```
handle, _ = pcap.OpenOffline("dump.pcap")  
defer handle.Close()
```

Creating a Packet Source

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

Read One Packet

```
packet, _ := packetSource.NextPacket()  
fmt.Println(packet)
```

Read All Packets

```
for packet := range packetSource.Packets() {  
    fmt.Println(packet)  
}
```

Capturing with Filter

```
handle.SetBPFFilter("tcp and port 80")
```

Berkeley Packet Filter Examples

- `10.1.1.3 # IP to and from`
- `128.3/16`
- `port 53`
- `host 8.8.8.8 and udp port 53`
- `net 199.16.156.0/22 and port 80`
- `(port 80 or port 443) and not host 192.168.0.1`

Opening pcap file for Writing

```
dumpFile, _ := os.Create("dump.pcap")
defer dumpFile.Close()

packetWriter := pcapgo.NewWriter(dumpFile)
packetWriter.WriteHeader(
    65535, // Snapshot length
    layers.LinkTypeEthernet,
)
```

Writing Pcap File

```
for packet := range packetSource.Packets() {  
  
    packetWriter.WritePacket(  
        packet.Metadata().CaptureInfo,  
        packet.Data(),  
    )  
  
}
```

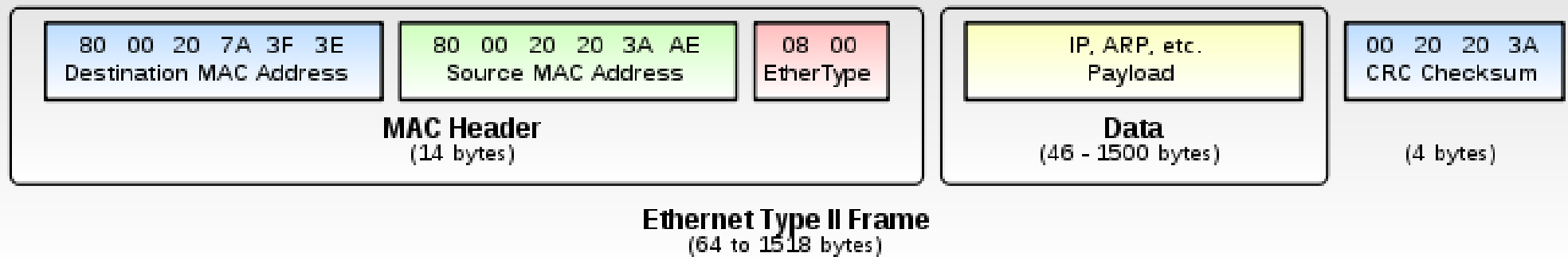
List Packet Layers

```
for _, layer := range packet.Layers() {  
    fmt.Println(layer.LayerType())  
}
```

Visualizing Packet Layers



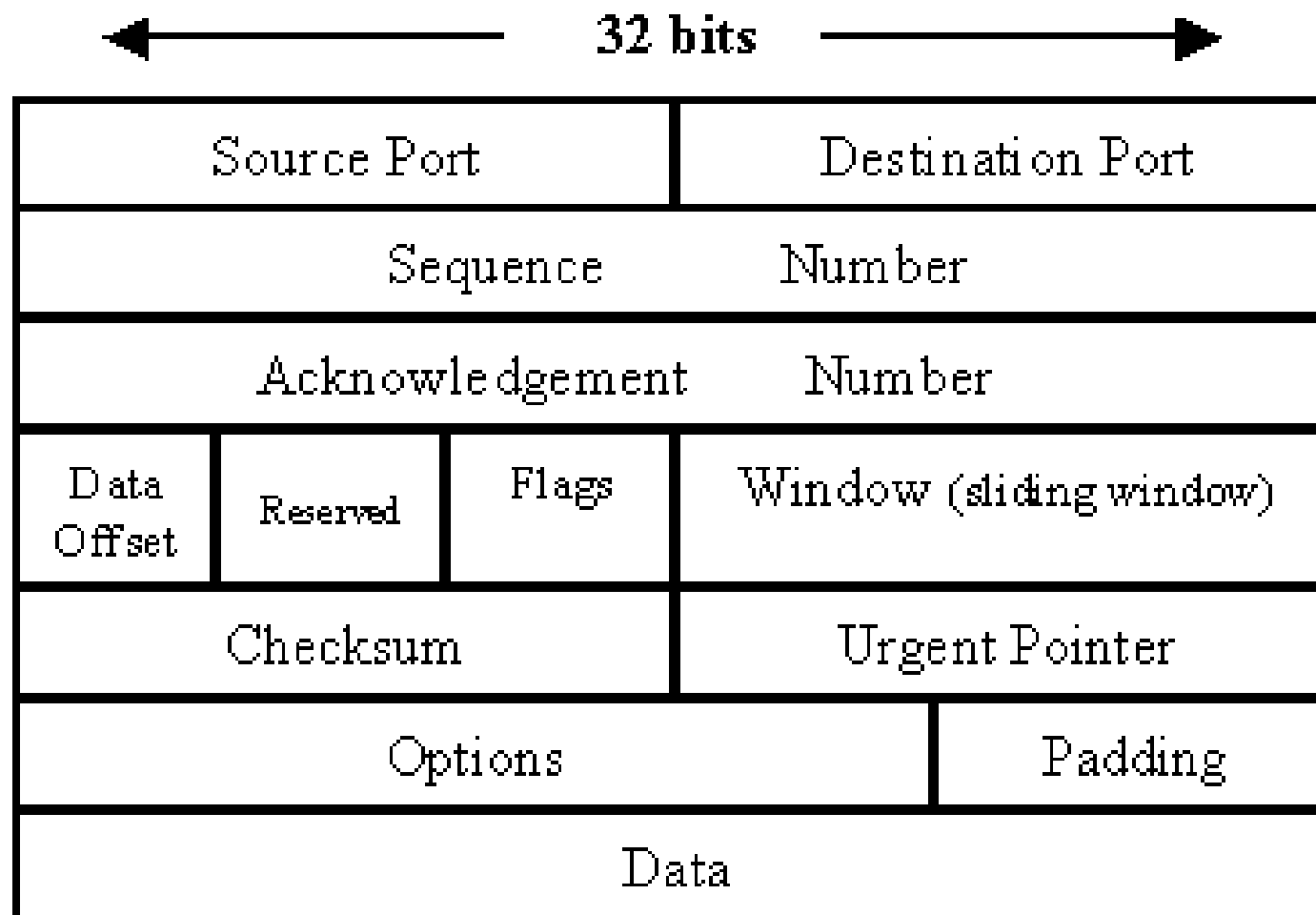
Ethernet Packet Structure



IP Packet Structure

0	4	8	16	19	31
Version	IHL	Type of Service	Total Length		
Identification			Flags	Fragment Offset	
Time To Live		Protocol	Header Checksum		
Source IP Address					
Destination IP Address					
Options					Padding

TCP Packet Structure



Analyze IPv4 Layer

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

```
if ipLayer != nil {
```

```
    ip, _ := ipLayer.(*layers.IPv4)
```

```
    fmt.Println(ip.SrcIP, ip.DstIP)
```

```
    fmt.Println(ip.Protocol) // TCP
```

```
}
```


Analyze TCP Layer

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

if tcpLayer != nil {
    tcp, _ := tcpLayer.(*layers.TCP)
    fmt.Println(tcp.SrcPort)
    fmt.Println(tcp.DstPort)
}
```

Decoding Packet Layers

```
ethernetPacket := gopacket.NewPacket(  
    packet, layers.LayerTypeEthernet, gopacket.Default)
```

```
ipPacket := gopacket.NewPacket(  
    packet, layers.LayerTypeIPv6, gopacket.NoCopy)
```

```
tcpPacket := gopacket.NewPacket(  
    packet, layers.LayerTypeTCP, gopacket.Lazy)
```

Faster Decoding (1/2)

```
// Create everything we'll be using
var eth layers.Ethernet
var ip4 layers.IPv4
var tcp layers.TCP
parser := gopacket.NewDecodingLayerParser(
    layers.LayerTypeEthernet, &eth, &ip4, &tcp)
decodedLayers := []gopacket.LayerType{}
```

Faster Decoding (2/2)

```
for packet := range packetSource.Packets() {  
  
    parser.DecodeLayers(packet, &decodedLayers)  
    for _, layerType := range decodedLayers {  
        fmt.Println(layerType)  
    }  
  
}
```

Other Supported Layers

- ARP
- CiscoDiscovery
- DHCP
- DNS
- Dot11
- ICMP
- PPPoE
- USB
- 118 registered layers in package

Common Packet Layers

	// Example
packet.LinkLayer()	// Ethernet
packet.NetworkLayer()	// Ipv4/6
packet.TransportLayer()	// TCP/UDP
packet.ApplicationLayer()	// HTTP
packet.ErrorLayer()	

Custom Layers (1/5)

```
// Register custom layer
var MyLayerType = gopacket.RegisterLayerType(
    12345,                // Unique ID
    "MyLayerType",       // Unique name
    gopacket.DecodeFunc(decodeMyLayer),
)
```

Custom Layers (2/5)

```
// Define the layer contents
type MyLayer struct {
    Header []byte
    payload []byte
}
```


Custom Layers (3/5)

```
// Define the decode function
func decodeMyLayer(
    data []byte,
    p gopacket.PacketBuilder) error
{
    p.AddLayer(&MyLayer{data[:4], data[4:]})
    return p.NextDecoder(layers.LayerTypeEthernet)
}
```

Custom Layers (4/5)

```
// Satisfy the function requirements
func (m MyLayer) LayerType() LayerType {
    return MyLayerType
}
func (m MyLayer) LayerContents() []byte {
    return m.Header
}
func (m MyLayer) LayerPayload() []byte {
    return m.payload
}
```

Custom Layers (5/5)

```
// Decode like any other layer
decodedPacket := gopacket.NewPacket(
    data,
    MyLayerType,
    gopacket.Default,
)
```

Creating Packets

```
buffer = gopacket.NewSerializeBuffer()  
options := gopacket.SerializeOptions{}  
gopacket.SerializeLayers(buffer, options,  
    &layers.Ethernet{},  
    &layers.IPv4{},  
    &layers.TCP{},  
    gopacket.Payload([]byte{65, 66, 67}),  
)
```

Sending Packet

```
handle.WritePacketData(buffer.Bytes())
```

Flow and Endpoint

```
someFlow := gopacket.NewFlow(  
    layers.NewUDPPortEndpoint(1000),  
    layers.NewUDPPortEndpoint(500))  
  
t := packet.NetworkLayer() // Check nil  
if t.TransportFlow() == someFlow {  
    fmt.Println("UDP 1000->500 found.")  
}
```

Project Ideas

- Detect blacklisted Ips
- Fuzz network services
- Monitor network traffic flow
- Port scanner
- Firewall
- IDS
- Reverse Engineering Mobile App APIs

Demos

- Questions before demos?