

# Lab 9: Network Capture

---

Link: <http://www.asecuritysite.com/csn09112/software01>

Video demo: <https://www.youtube.com/watch?v=raphJCH2SPE>

## 1 Number formats

---

Within cryptography we often have to present numbers in different formatting, and typically have to convert from decimal into hexadecimal (based 16). Enter the following Python program:

```
import sys
val=10

if (len(sys.argv)>1):
    val=int(sys.argv[1])

print "Hex: ",hex(val)
print "Decimal: ",val
print "Octal: ",oct(val)
print "Binary: ",bin(val)
```

Now use it to complete the following table:

Decimal	Hex	Octal	Binary
10			
	0x23		
		032	
			1111000

## 2 Capturing packets

---

We will use Wireshark and WinPCap fairly extensively through the module. You can download the WinPCap Python script [here](#). Put this into the default Python folder (such as c:\python27). Next create the following script:

```
## Based on code at https://code.google.com/p/winpcapy
import ctypes
from winpcapy import *
import time
import sys
import string

# Packet capture function
PHAND=CFUNCTYPE(None, POINTER(c_ubyte), POINTER(pcap_pkthdr), POINTER(c_ubyte))

## Callback function which is called for every new packet
def _packet_handler(param, header, pkt_data):
    local_tv_sec = header.contents.ts.tv_sec
    ltime=time.localtime(local_tv_sec);
    timestr=time.strftime("%H:%M:%S", ltime)
    print
    print("%s,%.6d len:%d" % (timestr, header.contents.ts.tv_usec, header.contents.len))

def get_ad():
    i=0
    d=alldevs.contents

    while d:
        i=i+1
        print("%d. %s" % (i, d.name))
        print (" (%s)\n" % (d.description))
        if d.next:
            d=d.next.contents
        else:
            d=False

    print ("Enter the interface number (1-%d):" % (i))
    inum= input('--> ')

    inum=int(inum)

    d=alldevs

    ## Get Selected adaptor
    for i in range(0,inum-1):
        d=d.contents.next
    return d.contents
```

```

## Define the callback function name
packet_handler=PHAND(_packet_handler)

alldevs=POINTER(pcap_if_t)()
errbuf= create_string_buffer(PCAP_ERRBUF_SIZE)

## Find all the devices
if (pcap_findalldevs(byref(alldevs), errbuf) == -1):
    print ("Error in pcap_findalldevs: %s\n" % errbuf.value)
    sys.exit(1)

## Get adaptor
d=get_ad()
adhandle = pcap_open_live(d.name,65536,1,1000,errbuf)

print("\nStarting to listen on %s...\n" % (d.description))

## Get 20 packets
pcap_loop(adhandle, 20, packet_handler, None)
pcap_close(adhandle)

```

Run the script. What are the names of your interfaces?

For the first 20 packets, what is the minimum and maximum packet size?

### 3 Displaying IP addresses

Next we will parse the packets for the IP addresses. First add the following to define the parsing of the packets:

```

u_short = c_ushort
u_char = c_ubyte
u_int = c_int

class ip_address(Structure):
    _fields_ = [("byte1", u_char),
                ("byte2", u_char),
                ("byte3", u_char),
                ("byte4", u_char)]

```

```

class ip_header(BigEndianStructure):
    _fields_ = [("ver_ihl", u_char),
                ("tos", u_char),
                ("tlen", u_short),
                ("identification", u_short),
                ("flags_fo", u_short),
                ("ttl", u_char),
                ("proto", u_char),
                ("crc", u_short),
                ("saddr", ip_address),
                ("daddr", ip_address),
                ("op_pad", u_int)]

```

Next replace the call back function with:

```

## Callback function which is called for every new packet
def _packet_handler(param,header,pkt_data):

    # retrieve the position of the ip header
    v_pkt_data = ctypes.cast(pkt_data, ctypes.c_void_p)
    v_ip_header = ctypes.c_void_p(v_pkt_data.value + 14)
    pih = ctypes.cast(v_ip_header, ctypes.POINTER(ip_header))
    ih = pih.contents
    print("{}.{}.{}.{} -> {}.{}.{}.{}".format(ih.saddr.byte1, ih.saddr.byte2, ih.saddr.byte3, ih.saddr.byte4, ih.daddr.byte1,
    ih.daddr.byte2, ih.daddr.byte3, ih.daddr.byte4))

```

Run the code and find the IP address connections for the first five connections?

## 4 Displaying connection details

Now we will read the TCP header, and which follows the IP address. In this case we will just display the TCP ports. First we add the format of the TCP packet (we have just used the first four fields):

```

class tcp_header(BigEndianStructure):
    _fields_ = [("source_port", u_short),
                ("destination_port", u_short),
                ("seq", u_int),
                ("ack", u_int)]

```

And we can replace the call back function:

```
def _packet_handler(param,header,pkt_data):  
    # retrieve the position of the ip header  
    v_pkt_data = ctypes.cast(pkt_data, ctypes.c_void_p)  
    v_ip_header = ctypes.c_void_p(v_pkt_data.value + 14)  
    pih = ctypes.cast(v_ip_header, ctypes.POINTER(ip_header))  
    ih = pih.contents  
  
    ip_len = (ih.ver_ihl & 0xf) * 4  
    th = ctypes.cast(ctypes.cast(pih, ctypes.c_void_p).value + ip_len,  
                    ctypes.POINTER(tcp_header)).contents  
  
    print("{}.{}.{}.{}:{}. -> {}.{}.{}.{}:{}".format(ih.saddr.byte1, ih.saddr.byte2, ih.saddr.byte3, ih.saddr.byte4,  
th.source_port,ih.daddr.byte1, ih.daddr.byte2, ih.daddr.byte3, ih.daddr.byte4,th.destination_port))
```

Run the code and find the IP address connections and TCP ports used for the first five packets?

## 5 Examining the Transport Layer protocol

The problem with the previous example is that there can be several transport layer protocols. So we must look at the Protocol field in the IP packet. Now modify your packet handler to add the IP Protocol field:

```
print("{}.{}.{}.{}:{}. -> {}.{}.{}.{}:{} Protocol: {}".format(ih.saddr.byte1, ih.saddr.byte2, ih.saddr.byte3,  
ih.saddr.byte4, th.source_port,ih.daddr.byte1, ih.daddr.byte2, ih.daddr.byte3, ih.daddr.byte4,th.destination_port,ih.proto))
```

Now run the Python program, and generate some traffic (such as loading a Web page. You will now see other protocols, such as 6- TCP and 17 - UDP. List the protocols that you see:

Run the code and find the IP address connections and TCP ports used for the first five packets?

## 6 Filtering for TCP

---

Now we can filter for just TCP traffic by examining the IP Protocol field. For this just replace your packet handler with:

```
def _packet_handler(param,header,pkt_data):
    # retrieve the position of the ip header
    v_pkt_data = ctypes.cast(pkt_data, ctypes.c_void_p)
    v_ip_header = ctypes.c_void_p(v_pkt_data.value + 14)
    pih = ctypes.cast(v_ip_header, ctypes.POINTER(ip_header))
    ih = pih.contents

    ip_len = (ih.ver_ihl & 0xf) * 4
    th = ctypes.cast(ctypes.cast(pih, ctypes.c_void_p).value + ip_len,
                    ctypes.POINTER(tcp_header)).contents
    if (ih.proto==6):
        print("TCP: {}.{}.{}.{}:{}.{} -> {}.{}.{}.{}:{}.{} Protocol: {}".format(ih.saddr.byte1, ih.saddr.byte2, ih.saddr.byte3,
ih.saddr.byte4, th.source_port,ih.daddr.byte1, ih.daddr.byte2, ih.daddr.byte3, ih.daddr.byte4,th.destination_port))
```

Next generate some traffic by accessing a Web site (or refreshing the cache). Note the IP addresses and TCP ports of the Web connections:

If you go to <https://Google.com>, and run your script, which server port is used?

If you go to <http://asecuritysite.com>, and run your script, which server port is used?

Why do the two sites differ in the server ports?

## 7 Tutorial

---

1. Most of the IP packets are IP Version 4. Read the IP Version number from the first four bits with:

```
ip_ver = (ih.ver_ihl & 0xf0) >> 4
print "IP version: ",ip_ver
```

Run the Python program and capture some traffic. Which is the version number defined in the packets, and what does the "& 0xf0" and ">> 4" parts of the code do?

2. In the previous Python program, the TCP fields have not been fully defined. Figure 1 shows the Ethernet, IP and TCP fields. Using the TCP header definition, update the TCP class definition in your Python program to include all of the fields:

```
class tcp_header(BigEndianStructure):
    _fields_ = [("source_port", u_short),
                ("destination_port", u_short),
                ("seq", u_int),
                ("ack", u_int),
                ("flags", u_short),
                ("window", u_short),
                ("checksum", u_short),
                ("urgent", u_short),
                ("options", u_int)]
```

Test the output. Do the SEQ and ACK tie-up on a connection?

2.1 What values do you get for the Flags field?

2.2 Now mask off the flags with:  
`print "Flags: ", th.flags & 0x0ff,`

2.3 What values do you know get? Can you match them to the TCP flags?



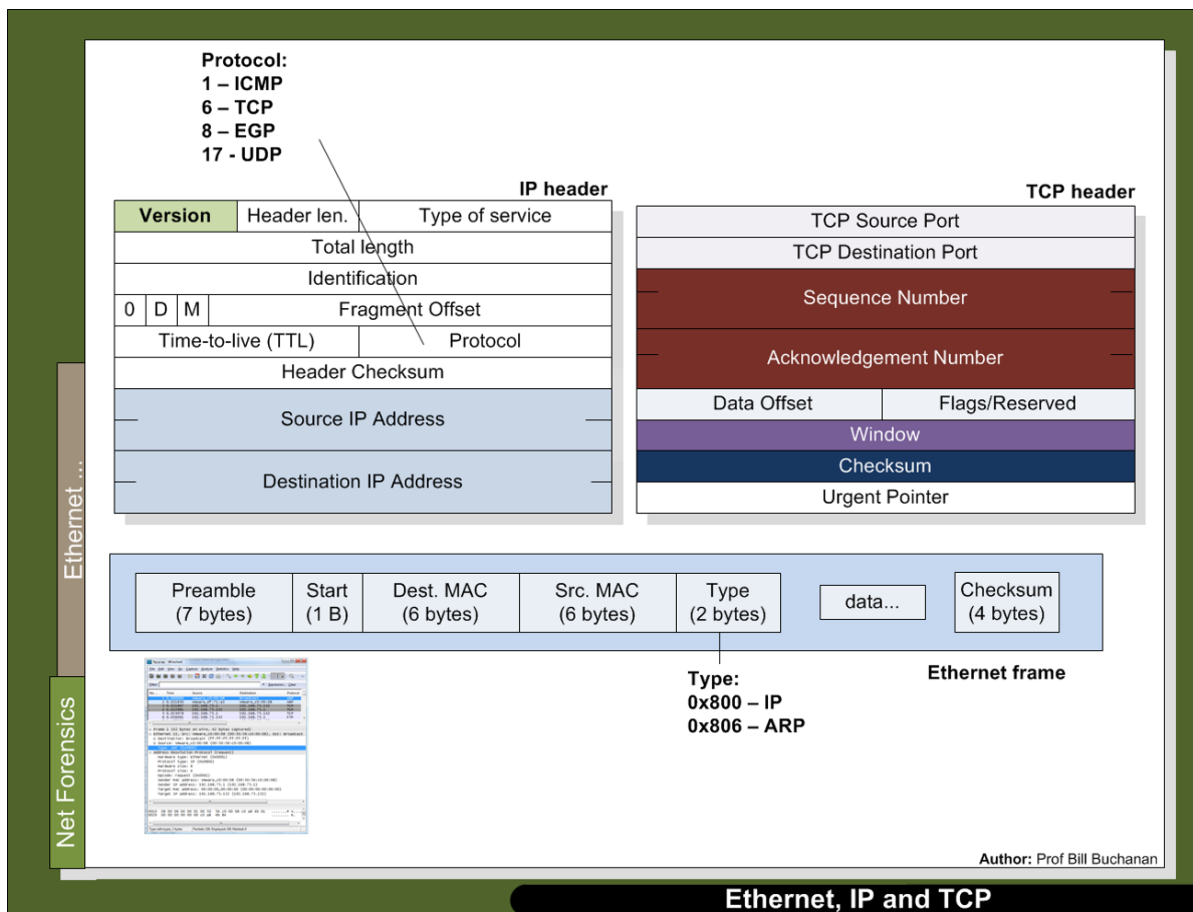


Figure 1: Ethernet, IP and TCP

3. We will now put the program into an infinite loop and break when there is a keypress. For this we use the Pywin32 library, which contains the pyHook class:

<http://sourceforge.net/projects/pywin32/files/pywin32/Build%20219/pywin32-219.win32-py2.7.exe/download>

Next replace:

```
## Get 20 packets
pcap_loop(adhandle, 20, packet_handler, None)
pcap_close(adhandle)
```

with:

```
import pyHook,pythoncom

def OnKeyboardEvent(event):
    exit()
    pcap_breakloop(ahandle)
    pcap_close(adhandle)

hm = pyHook.HookManager()
hm.KeyDown = OnKeyboardEvent
hm.HookKeyboard()

while True:
    try:
        while True:
            pcap_loop(adhandle, 1, packet_handler, None)
            pythoncom.PumpWaitingMessages()
    except KeyboardInterrupt:
        exit()
```

You should now be able to capture until a key is pressed.

## 8 Appendix

---

The files for Part 2 are:

- Part 1. Here.
- Part 2. Here.
- Part 3. Here.
- Part 4. Here.
- Final solution. Here.

And you'll need to download the following:

<https://asecuritysite.com/winpcapy.zip>