Warning - Don't get fired!

Some people consider Wireshark an "offensive weapon" in the area of network security. They will not want it hooked up to their network. Depending on policy, at a workplace or school it could lead to disciplinary action.

And I just told you to feel free to download Wireshark on your own laptop.

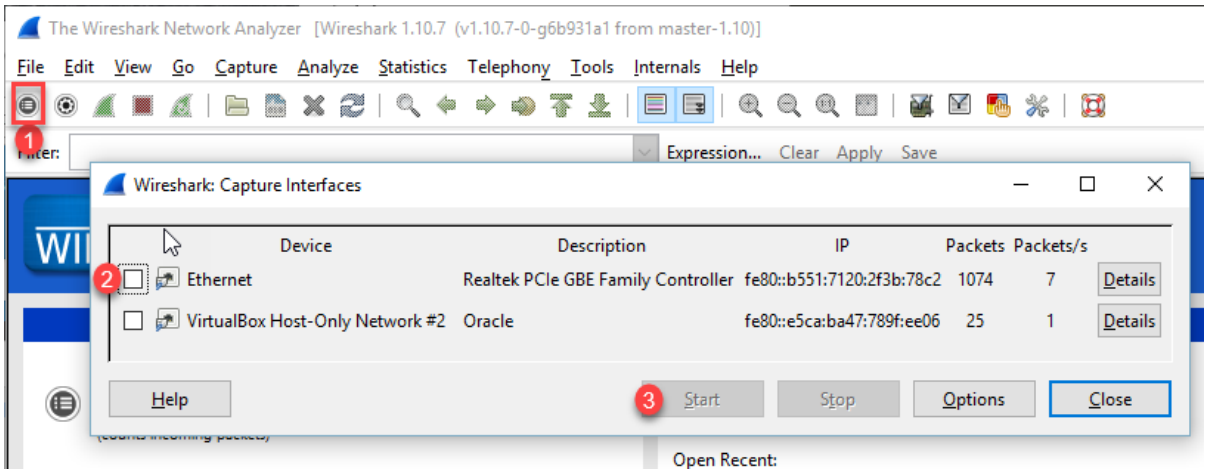
There is a lot of misconceptions around what Wireshark can and can't do. Even amongst people who consider themselves experts. It won't make arguing your case any easier if you do get into trouble.

In the case of Simpson, the administration currently does not want Wireshark on the campus network. Keep to using our locally-created networks created just for this class that you can happily mess up all you want.

So feel free to download Wireshark to your laptop. Unless you want to hook it up to Simpson's network.

Start capture

The button to start capture isn't very obvious. You also need to select the network adapter to capture:



After the capture, you might have too many packets. You can filter the packets with a special Wireshark filter language.

Sending Raw Ethernet Tutorial

Here is C code that will send raw Ethernet packets:

```
/*
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
 *
 * Original version from Austin Martin:
 * https://gist.github.com/austinmarton/1922600
 *
 * Adapted by Paul Craven
 */

#include <arpa/inet.h>
#include <linux/if_packet.h>
#include <stdio.h>
#include <string.h>
```

```c
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/ether.h>
#include <unistd.h>

/* Replace the bytes below with the MAC address
   you are sending to.
   So if 'ifconfig' on the RECEIVER (not the sender) says:
   wlan0     Link encap:Ethernet  HWaddr b8:27:eb:44:08:62
   You'd replace the bytes below with:
   #define DEST_MAC0 0xB8
   #define DEST_MAC1 0x27
   #define DEST_MAC2 0xEB
   #define DEST_MAC3 0x44
   #define DEST_MAC4 0x08
   #define DEST_MAC5 0x62
*/

#define DEST_MAC0    0xB8
#define DEST_MAC1    0x27
#define DEST_MAC2    0xEB
#define DEST_MAC3    0x44
#define DEST_MAC4    0x08
#define DEST_MAC5    0x62

#define DEFAULT_IF  "wlan0"
#define BUF_SIZ     1024

int main(int argc, char *argv[])
{
    int sockfd;
    int i;
    struct ifreq if_idx;
    struct ifreq if_mac;
    int tx_len;
    char sendbuf[BUF_SIZ];
    struct sockaddr_ll socket_address;
    char ifName[IFNAMSIZ];

    /* Get interface name */
    if (argc > 1)
        strcpy(ifName, argv[1]);
    else
        strcpy(ifName, DEFAULT_IF);

    /* Open RAW socket to send on */
    if ((sockfd = socket(AF_PACKET, SOCK_RAW, IPPROTO_RAW)) == -1) {
        perror("socket");
    }

    /* Get the index of the interface to send on */
    memset(&if_idx, 0, sizeof(struct ifreq));
    strncpy(if_idx.ifr_name, ifName, IFNAMSIZ-1);
    if (ioctl(sockfd, SIOCGIFINDEX, &if_idx) < 0)
        perror("SIOCGIFINDEX");

```

```
    /* Get the MAC address of the interface to send on */
    memset(&if_mac, 0, sizeof(struct ifreq));
    strncpy(if_mac.ifr_name, ifName, IFNAMSIZ-1);
    if (ioctl(sockfd, SIOCGIFHWADDR, &if_mac) < 0)
        perror("SIOCGIFHWADDR");

    // Loop forever
    while(1) {


        /* Buffer of BUF_SIZ bytes we'll construct our frame in.
           First, clear it all to zero. */
        memset(sendbuf, 0, BUF_SIZ);
        tx_len = 0;

        /* Construct the Ethernet header */

        /* Ethernet header */
        /* Destination address */
        sendbuf[tx_len++] = DEST_MAC0;
        sendbuf[tx_len++] = DEST_MAC1;
        sendbuf[tx_len++] = DEST_MAC2;
        sendbuf[tx_len++] = DEST_MAC3;
        sendbuf[tx_len++] = DEST_MAC4;
        sendbuf[tx_len++] = DEST_MAC5;

        /* Create the source */
        sendbuf[tx_len++] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[0];
        sendbuf[tx_len++] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[1];
        sendbuf[tx_len++] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[2];
        sendbuf[tx_len++] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[3];
        sendbuf[tx_len++] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[4];
        sendbuf[tx_len++] = ((uint8_t *)&if_mac.ifr_hwaddr.sa_data)[5];

        /* Ethertype field */
        sendbuf[tx_len++] = 0x08;
        sendbuf[tx_len++] = 0x00;

        /*
        Packet data
        This is the 'payload'. Replace this with your real data.
        Because you'll probably have more interesting things to send
        than the hex 0xDEAD 0xBEEF
        */
        sendbuf[tx_len++] = 0xde;
        sendbuf[tx_len++] = 0xad;
        sendbuf[tx_len++] = 0xbe;
        sendbuf[tx_len++] = 0xef;

        /* Index of the network device */
        socket_address.sll_ifindex = if_idx.ifr_ifindex;
        /* Address length*/
        socket_address.sll_halen = ETH_ALEN;
        /* Destination MAC */
        socket_address.sll_addr[0] = DEST_MAC0;
        socket_address.sll_addr[1] = DEST_MAC1;
        socket_address.sll_addr[2] = DEST_MAC2;
        socket_address.sll_addr[3] = DEST_MAC3;
```

```
        socket_address.sll_addr[4] = DEST_MAC4;
        socket_address.sll_addr[5] = DEST_MAC5;

        /* Send packet */
        if (sendto(sockfd, sendbuf, tx_len, 0, (struct sockaddr*)&socket_address,␣
↪sizeof(struct sockaddr_ll)) < 0)
            printf("Send failed\n");
        else {
            printf("Sent :");
            for (i=0; i < tx_len; i++)
                printf("%02x:", sendbuf[i]);
            printf("\n");
        }
        /* Wait specified number of microseconds
           1,000,000 microseconds = 1 second
           */
        usleep(1000000);
    }
    return 0;
}
```

How do you run this? Save it in a file on your computer. C files should end in `.c`. Let's call this file `send.c`.

Now we have to compile it to a computer program. We will use the Gnu C Compiler. That program is available on the command line of your Raspberry Pi with `gcc`

You can compile the code with:

```
gcc send.c
```

But wait! That will make the output of the compile have a default name of `a.out`. That isn't a good name. Instead let's use:

```
gcc send.c -o send
```

This will use the Gnu C Compiler `gcc` to compile the C program `send.c` and save the output into a file named `send`. (Without the `.c` at the end.)

Now we need to run the program. C compiles into the native language. We don't need a Java run time, or Python, or whatever. We can just run it.

You can the program with:

```
send
```

Ah, but wait! We need admin privileges to do something so low-level on the network. So you might want do use:

```
sudo send
```

But that doesn't work either. Because by default the admin user only runs command from directories we know are 'safe.' We need to tell the computer it is ok to run the command from *this* directory.

You can run the program with:

```
sudo ./send
```

Want to receive? Here is a program to receive packets:

```
/*
 *  This program is free software: you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation, either version 3 of the License, or
 *  (at your option) any later version.
```

```
 *
 * Original version from Austin Martin:
 * https://gist.github.com/austinmarton/2862515
 *
 * Adapted by Paul Craven
 */

#include <arpa/inet.h>
#include <linux/if_packet.h>
#include <linux/ip.h>
#include <linux/udp.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/ether.h>

/* Replace this with this computer's MAC address
   In this case, the MAC address is b8:27:eb:44:08:62.
   You need to update this.
*/
#define DEST_MAC0 0xB8
#define DEST_MAC1 0x27
#define DEST_MAC2 0xEB
#define DEST_MAC3 0x44
#define DEST_MAC4 0x08
#define DEST_MAC5 0x62

#define ETHER_TYPE  0x0800

/* Change this to your interface name.
   Such as "wlan0" or "eth0".
   This program seems to ignore this parameter, and
   listens to everything.
   */

#define DEFAULT_IF  "eth0"
#define BUF_SIZ     1024

int main(int argc, char *argv[])
{
    int sockfd, i;
    int sockopt;
    ssize_t numbytes;
    struct ifreq ifopts;    /* set promiscuous mode */
    uint8_t buf[BUF_SIZ];
    char ifName[IFNAMSIZ];

    /* --- Get ready to listen --- */

    /* Copy the interface name to a character buffer */
    strcpy(ifName, DEFAULT_IF);

    /* Open PF_PACKET socket, listening for EtherType ETHER_TYPE */
    if ((sockfd = socket(PF_PACKET, SOCK_RAW, htons(ETHER_TYPE))) == -1) {
        perror("listener: socket");
```

```
64          return -1;
65      }
66
67      /* Set interface to promiscuous mode - do we need to do this every time? */
68      strncpy(ifopts.ifr_name, ifName, IFNAMSIZ-1);
69      ioctl(sockfd, SIOCGIFFLAGS, &ifopts);
70      ifopts.ifr_flags |= IFF_PROMISC;
71      ioctl(sockfd, SIOCSIFFLAGS, &ifopts);
72
73      /* Allow the socket to be reused - in case connection is closed prematurely */
74      if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &sockopt, sizeof sockopt) == -1)
↪{
75          perror("setsockopt");
76          close(sockfd);
77          exit(EXIT_FAILURE);
78      }
79      /* Bind to device */
80      if (setsockopt(sockfd, SOL_SOCKET, SO_BINDTODEVICE, ifName, IFNAMSIZ-1) == -1)  {
81          perror("SO_BINDTODEVICE");
82          close(sockfd);
83          exit(EXIT_FAILURE);
84      }
85
86      /* --- Main loop to listen, and print a packet --- */
87
88      // Loop forever
89      while(1) {
90
91          /* Receive the data */
92
93          //printf("Listener: Waiting to receive...\n");
94          numbytes = recvfrom(sockfd, buf, BUF_SIZ, 0, NULL, NULL);
95          //printf("Listener: got packet %lu bytes\n", numbytes);
96
97          /* Check the packet is for me
98             by looking at the destination address. */
99          if (buf[0] == DEST_MAC0 &&
100             buf[1] == DEST_MAC1 &&
101             buf[2] == DEST_MAC2 &&
102             buf[3] == DEST_MAC3 &&
103             buf[4] == DEST_MAC4 &&
104             buf[5] == DEST_MAC5) {
105
106             // printf("Correct destination MAC address\n");
107
108             if(buf[14] == 0x45) {
109                 /* This is likely just an SSH packet from a
110                    remove SSH terminal. Let's ignore those
111                    and not do anything here.
112                 */
113
114             } else {
115                 /* We've got data!
116                    Print the packet */
117                 printf("Data:");
118                 for (i=0; i<numbytes; i++)
119                     printf("%02x:", buf[i]);
120                 printf("\n");
```

```
            }

        } else {
            /* Ok, this data was not intended for us.
                   Which means we plugged in the wrong MAC
               Or it was sent to ff:ff:ff:ff:ff:ff which is
                   broadcast for 'everyone'

            printf("Wrong destination MAC: %x:%x:%x:%x:%x:%x\n",
                            buf[0],
                            buf[1],
                            buf[2],
                            buf[3],
                            buf[4],
                            buf[5]);
            */
        }

    }
    close(sockfd);
    return 0;
}
```

Network Layer

Send and Receive Datagrams

Sending a datagram is easy. Below is Python code to do it:

Send Datagram

```
# Import the built-in library that manages network sockets
import socket

# Address of the target. Replace this with the address that you want.
destination_ip_address = '127.0.0.1'

# Port to send the packet to.
destination_ip_port = 10000

# Enter the data content of the UDP packet as an array of
# bytes. That's why there is a 'b' in front of the string.
packet_data = b'This is a test message.'

# Initialize a network socket
# SOCK_DGRAM specifies that this is UDP
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, 0)

# Connect the socket. For UDP this doesn't send anything yet. It does say
# that when we do send data, where will it go.
s.connect((destination_ip_address, destination_ip_port))

# Send the data.
```