Getting ready to Sniff
Part I
ECE-C433 Network Programming 2005-2006
Drexel University
Presented by Sukrit Dasgupta, 30th May 2006
Perennially complaining TA

- Sleep ~ negligible
- Food = 0
- Tired = infinity
- Errors = plenty
- Ask me questions !!
So far ...

• Increment server
• Email client
• (Hopefully) webserver

• All work is based in the Application Layer
• What happens after that ??
Outline

• What and why?
• How?
• Libraries and pre-requisites
• Finding your devices and networks
• Capturing your first packet
• Starting a capture session
• A cool and simple usage to whet your appetite
• Summary
What and Why?

- Grabbing packets off a channel (shared or unshared)
- Analyzing grabbed packets
- Observe and study network behavior
- Testing custom built network apps
- Potentially malicious / ‘Fun’ uses
  - Sniffing passwords/logins
  - Observing user preferences
  - 1000’s of other things ...
How?

High Level language + Libraries

C/C++/Java

Linux/Mac OS/Windoze

LAN Card

Drexel

Smart User

eth0

eth1

ethX

NETWORK

Drexel
Libraries and pre-requisites

- We’ll use ‘C’
  - Portable Packet capture library: libpcap
    Header: `pcap.h OR winpcap.h`
  - Other common headers:
    - `sys/socket.h`
    - `netinet/in.h`
    - `arpa/inet.h`
- Root access for running
- Ethernet packets ONLY ! (for the time being ;-)

pcap_lookupdev(…)

• Finds the first usable Ethernet device
• Returns: char*
  • Device name in string format: eth0
• Parameter: char *errbuf
  • Filled with error message or NULL
• device=pcap_lookupdev(errbuf)
pcap_lookupnet(…)

- Used to determining network number and mask associated with device
- Returns int
- Parameters
  - `const char *device`: the device detected
  - `bpf_u_int32 *netp`: network address
  - `bpf_u_int32 *maskp`: network mask
  - `char *errbuf`: error message, just in case
pcap_lookupnet(...) contd.

- `bpf_u_int32`: 32 bit unsigned integer
  - `bpf`: BSD Packet Filter
- `return=pcap_lookupnet(dev,netp,maskp,erbuff)`
- After execution, `netp` and `maskp` contain the corresponding addresses
- Need to be changed to readable form using `inet_ntoa(struct in_addr address)`
- `address.s_addr=netp`
- `address.s_addr=maskp`
Lets play ...

- `dev=pcap_lookupdev(errbuf);`
- `printf(dev);`
- `return=pcap_lookupnet(dev,netp,maskp,errbuf);`
- `address.s_addr=netp`
- `printf("%s",inet_ntoa(address));`
Towards the first capture

- Packet capture descriptor
  - Type: `pcap_t*` : similar to sockfd

- Packet Header
  - Type: `struct pcap_pkthdr*`
    - `struct timeval ts`
      - `long int tv_sec`: seconds
      - `long int tv_usec`: micro-seconds
    - `bpf_u_int32 caplen`: captured length
    - `bpf_u_int32 len`: length of packet
pcap_open_live(...)

- Returns the packet capture descriptor pcap_t*
- Parameters
  - `const char *device`: device to capture from
  - `int snaplen`: length to capture
  - `int promisc`: set promiscuous mode
  - `int to_ms`: timeout to wait for a packet
  - `char *errbuf`: error message just in case
- `descr=pcap_open_live(device,BUFSIZ,0,2,errbuf)`
pcap_next(…) 

- To get the next packet from the descriptor
- Returns: u_char*
- Parameters:
  - pcap_t *descriptor : capture descriptor
  - pcap_pkthdr *hdr : packet header

- packet = pcap_next(descriptor, hdr);
What is captured

Application Layer Data
Transport Layer Data
Network Layer Data
Link Layer Data
Lets play again ...

- `pcap_t *descriptor;`
- `struct pcap_pkthdr hdr;`
- `u_char *packet;`
- `descriptor=pcap_open_live(dev,BUFSIZ,0,2,err);`
- `packet=pcap_next(descriptor, hdr);`
- `printf("%d",hdr.len);`
- `printf("%lu",hdr->ts.tv_sec);`
Starting a capture session

• Need to loop and grab packets till a certain condition is met

• Two simple functions:
  • pcap_loop(…)
  • Your own function, called CALLBACK Fn.

• pcap_loop(…) captures a packet

• forwards the captured packet to your function for processing
pcap_loop(...)

- Returns int
- Parameters
  - pcap_t *descriptor: capture descriptor
  - int count: number of packets processed
  - pcap_handler callback: name of your function
  - u_char *user: parameters for your function
- pcap_loop(descriptor, 10, my_func, NULL);
Callback function

- Functionality is completely dependent on the programmer
- Returns: Any datatype
- Parameters: (Mandatory)
  - `u_char *userdefined`: coder’s parameters
  - `struct pcap_pkthdr *hdr`: packet header
  - `u_char *packet`: pointer to the packet data
my_callback()

- void my_callback(...)
- printf("%d packets received", count);
- count++;
- pcap_loop(descriptor, 2, my_callback, NULL);
Let’s play ...

- device=pcap_lookupdev(errbuf);
- descriptor=pcap_open_live(dev,BUFSIZE,0,5,err);
- pcap_loop(descriptor,10,my_callback,NULL);
Fun: Packet Arrival Analysis

• Grab every packet coming your way
• Retrieve the timestamp
• Let's see what happens over a long duration
What is known ...  
The Willinger, Taqqu et al. paper
What we find out ...

\[ x\text{-unit} = 1\text{ second} \]
\[ y\text{-unit} = \text{pkts}/x\text{-unit} \]

\[ x\text{-unit} = 0.1\text{ second} \]
\[ y\text{-unit} = \text{pkts}/x\text{-unit} \]

\[ x\text{ unit} = 0.01\text{ second} \]
\[ y\text{ unit} = \text{pkts}/x\text{ unit} \]
x-unit = 1 second  
y-unit = pkts/x-unit

x-unit = 0.1 second  
y-unit = pkts/x-unit

x unit = 0.01 second  
y unit = pkts/x unit
Summarizing

- Getting hold of our device for sniffing
- Obtaining the network mask and network number
- Capturing our first packet
- Looping the packet capture code
- Doing custom analysis
- Getting something useful out of something SO ELEMENTARY
Reasons to attend next class

• Applying filters to our packet capture engine
  • TCP, UDP, ARP packets etc
• Digging into the packets
  • Source IP, destination IP, port numbers etc
  • Analyzing Drexel user preferences
• More importantly
  • QUIZ !
THANKS!
Questions?