Cool Sniffing ...

Part 2

ECE-C433 Network Programming 2005-2006
Drexel University
Presented by Sukrit Dasgupta, 1st June 2006
What we did ...

• Basics of the libpcap library
• Finding your device
• Setting it up for capture (descriptor)
• Finding your network
• Capturing packets
• Something useful using the above ...
Mandatory whining
What we’ll TRY today ...

• Why we don’t need every packet
• What and how of filters
• Networking 101
• Ethernet Header
• Grabbing Ethernet header specific data
• IP Header
• Grabbing IP header specific data
• Putting it all together
Grabbing more than “every” ...

• We don't need every packet
  • In fact, we almost don't need 90% of the packets

• Interest depends on what we are looking for

• Need to “filter” out the not-needed packets

• pcap provides this filter
Coming up with a filter

• Syntax is simple yet powerful

  • QUALIFIERS
    • **type**: host, net and port : is a qualifier
    • **dir**: src, dst, src *or* dst, src *and* dst : direction
    • **proto**: ether, ip, arp, tcp, udp : any ‘layer’ protocol

• **src net 129.25 port 80**
  • Captures all HTTP data from the Drexel network
  • How do you know this ?
Fancier Filtering

• Using boolean operators
  • and, or, not

• To access data inside the packet
  • proto [ expr : size ]
    • proto represents the protocol layer: tcp, udp, ip, ethernet
    • expr : byte offset in the protocol of interest: AN INDEX
    • size : number of bytes in the field of interest (1, 2 or 4)
So ...

- src net 129.25 and port 80 and tcp[13] & 3 ! = 0
- All HTTP connections out of Drexel
- Only TCP packets with SYN or FIN
- tcp[13] ???
- tcp[13] & 3 ! = 0 ???
- 01 = FIN, 10 = SYN and 11 = 3 in binary
- 01 & 11 ! = 0 and 10 & 11 ! = 0
- Cool !! but what is TCP[13] !!!
Now, the code ...

- Compile a filter to be used
  - `pcap_compile(...)`
- Set the compiled filter
  - `pcap_setfilter(...)`

- THATS IT!
pcap_compile(...)

• Returns **int**

• **Parameters**
  • `pcap_t *descriptor`
  • `struct bpf_program *compiled`
  • `char *filter_string`
  • `int optimize`
  • `bpf_u_int32 netmask`
pcap_setfilter(...)  

- Returns `int`
- Parameters
  - `pcap_t *descriptor`
  - `struct bpf_program *compiled`
Replay ...

- `device=pcap_lookupdev(errbuf);`
- `descriptor=pcap_open_live(dev,BUFSIZE,0,5,err);`
- `pcap_lookupnet(dev,&net,&mask,errbuf);`
- `pcap_compile(descriptor,&compile,string,0,netp);`
- `pcap_setfilter(descriptor,&compile);`
- `pcap_loop(... stuff from the last class ... );`
Networking 101

• What is a MAC address? Which layer?
• What is a port? What layer has it/needs it?
• What is an IP address? Which layer?
IP ROUTING
MAC ADDRESS
MAC ADDRESS
MAC ADDRESS
MAC ADDRESS
MAC ADDRESS
MAC ADDRESS
Packet Structure

- Application Layer
- Transport Layer
- Network Layer
- Link Layer

Starting Address

IN MEMORY

Ethernet Header | IP Header | TCP/UDP Header | Data

Offset to get to IP? to TCP?
Typecasting

• `variable_typeA = (type A*) (some memory address)`
• `variable_typeA = memory address of some type A`?
• `ether_header_var = (ether_header*) blob`?
Ethernet Frame

Transmission order: left-to-right, bit serial

FCS error detection coverage

FCS generation span

<table>
<thead>
<tr>
<th>PRE</th>
<th>SFD</th>
<th>DA</th>
<th>SA</th>
<th>Length/Type</th>
<th>Data</th>
<th>Pad</th>
<th>FCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>46-1500</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

Field length in bytes

PRE = Preamble
SFD = Start-of-frame delimiter
DA = Destination address
SA = Source address
FCS = Frame check sequence
Ethernet Header

- Destination Address: 6 bytes
- Source Address: 6 bytes
- Type of packet: 2 bytes

```c
struct ether_header {
    u_char ether_dhost[ETHER_ADDR_LEN];
    u_char ether_shost[ETHER_ADDR_LEN];
    u_short ether_type;
}
```
Ethernet: Constants we need

• Source/Destination addresses

```c
struct ether_addr
{
    u_char octet[ETHER_ADDR_LEN];
}
```

• Types

  • `#define ETHERTYPE_IP   0x0800`
  • `#define ETHERTYPE_ARP  0x0806`
  • `#define ETHERTYPE_RARP 0x8035`
Ethernet specific code

- Typecast the “blob” of packet
  - `eth_ptr=(struct ether_header *)packet;`

- Get the source MAC address
  - `ether_nota((struct ether_addr*)eth_ptr->ether_shost);`

- Get the destination MAC address
  - `ether_nota((struct ether_addr*)eth_ptr->ether_dhost);`

- Get the type of ethernet packet
  - `ether_type=ntohs(eptr->ether_type);`
Lets play ...

- `device=pcap_lookupdev(errbuf);`
- `descriptor=pcap_open_live(dev,BUFSIZE,0,5,err);`
- `pcap_loop(descriptor,10,eth_code,NUL);`
- `handle_ethernet(...)`
  - `eth_ptr=(struct ether_header*)packet;`
  - `ether_nota( (struct ether_addr*)eth_ptr->ether_shost );`
  - `ether_nota( (struct ether_addr*)eth_ptr->ether_dhost );`
  - `ether_type=ntohs(eptr->ether_type);`
IP Header

- Version
- IHL
- Type-of-service
- Total length
  - Identification
  - Flags
  - Fragment offset
- Time-to-live
- Protocol
- Header checksum
- Source address
- Destination address
- Options (+ padding)
- Data (variable)
struct ip{
    u_int8_t ip_vhl; /* header length, version */
#define IP_V(ip)(((ip)->ip_vhl & 0xf0) >> 4)
#define IP_HL(ip) ((ip)->ip_vhl & 0x0f)
    u_int8_t ip_tos; /* type of service */
    u_int16_t ip_len; /* total length */
    u_int16_t ip_id; /* identification */
    u_int16_t ip_off; /* fragment offset field */
#define IP_DF 0x4000 /* dont fragment flag */
#define IP_MF 0x2000 /* more fragments flag */
#define IP_OFFMASK 0x1fff /* mask for fragmenting bits */
    u_int8_t ip_ttl; /* time to live */
    u_int8_t ip_p; /* protocol */
    u_int16_t ip_sum; /* checksum */
    struct in_addr ip Src,ip_dst; /* source and dest address */
};
Lets just play ... :-( 
Putting it all together

- Apply a filter
- Grab only the IP packets
- Look at the destination
- Make a list or a database over time
- What do you get?
Summary

- 2 days of extreme tolerance (for you)
- 2 sleepless nights (for me)
- Lines of code that no one understands
- Lots of packets grabbed
- Some analyzed in great detail
- A new look to network programming
- An extremely difficult quiz coming up!
References

- tcpdump.org
- sniffit
- sniffex
- winpcap.org
- 10000s of other sites and docs on the Internet