COSC 458-647
Application Software Security

Tonight

- Short class …
- Intro to packet sniffing with libpcap
- Lab: Task 1 (problems 1 – 5)

Packet Sniffing with PCAP

Tutorial by Tim Casterns

Programming with PCAP

1. We begin by determining which interface we want to sniff on.
   - In Linux this may be something like eth0, in BSD it may be xl1, etc.
   - We can either define this device in a string, or we can ask pcap to provide us with the name of an interface that will do the job.

2. Initialize pcap.
   - We can sniff on multiple devices.
   - How do we differentiate between them?
     - Using file handles.
     - We must name our sniffing "session" so we can tell it apart from other such sessions.

Programming with PCAP

- Create a rule set, "compile" it, and apply it.
  - To sniff specific traffic (e.g.: only TCP/IP packets, only packets going to port 23, etc)
- Finally, we tell pcap to enter it's primary execution loop.
  - In this state, pcap waits until it has received however many packets we want it to.
  - Every time it gets a new packet in, it calls another function that we have already defined. The function that it calls can do anything we want; it can dissect the packet and print it to the user, it can save it in a file, or it can do nothing at all.
- After our sniffing needs are satisfied, we close our session and are complete.

Setting the device

```
#include <stdio.h>
#include <pcap.h>
int main(int argc, char *argv[]) {
   char *dev, errbuf[PCAP_ERRBUF_SIZE];
   dev = pcap_lookupdev(errbuf);
   if (dev == NULL) {
     fprintf(stderr, "Couldn't find default device: %s\n",errbuf);
     return(2);
   }
   printf("Device: %s\n", dev);
   return(0);
}
```

Opening the device for sniffing

The task of creating a sniffing session is really quite simple using

pcap_open_live()

```
pcap_t *pcap_open_live (
            char *device, // the device for sniffing
            int snaplen,  // max. # of bytes to sniff
            int promisc,  // promiscuous mode
            int to_ms,    // time out (in milliseconds)
            char *ebuf    // the error buffer
)
```

The function returns the session handler.

Opening the device for sniffing (cont'd)

Example:

```
#include <pcap.h>
...
pcap_t *handle;
handle = pcap_open_live(somedev, BUFSIZ, 1, 1000, errbuf);
if (handle == NULL) {
 fprintf(stderr, "Couldn't open device %s: %s\n", somedev,
errbuf);
 return(2);
}
```

- This code fragment opens the device stored in the strong "somedev", tells it to read however many bytes are specified in BUFSIZ (which is defined in pcap.h).
- We are telling it to put the device into promiscuous mode, to sniff until an error occurs, and if there is an error, store it in the string errbuf; it uses that string to print an error message.

Promiscuous vs. Non-promiscuous sniffing

- In standard, non-promiscuous sniffing, a host is sniffing only traffic that is directly related to it.
  - Only traffic to, from, or routed through the host will be picked up by the sniffer.

- Promiscuous mode, on the other hand, sniffs all traffic on the wire.
  - Advantage: Provides more packets for sniffing
  - Disavantage:
    - Promiscuous mode sniffing is detectable; a host can test with strong reliability to determine if another host is doing promiscuous sniffing.
    - Second, it only works in a non-switched environment (such as a hub, or a switch that is being ARP flooded).
    - Third, on high traffic networks, the host can become quite taxed for system resources.

Filtering traffic: pcap_compile() & pcap_setfilter()

- We may only be interested in specific traffic.
  - For instance, to sniff on port 23 (telnet) in search of passwords, or to highjack a file being sent over port 21 (FTP), or DNS traffic (port 53 UDP).
- Before applying our filter, we must "compile" it with pcap_compile()
  - The filter expression is a regular string (char array).

```
int pcap_compile(
  pcap_t *p,                  // the session handle
  struct bpf_program *fp,     // complied filter
  char *str,                  // the expression
  int optimize,               // optimize or not?
  bpf_u_int32 netmask         //mask of the network
                              // the filter applies to.
)
```

Filtering traffic: pcap_compile() & pcap_setfilter()

- After the expression has been compiled, it is time to apply it with

pcap_setfilter()

```
int pcap_setfilter(
   pcap_t *p,                    // session handler
   struct bpf_program *fp // the compiled version of the expression
)
```

Callback functions `pcap_loop()`

- Just like hook functions in netfiler.

```
int pcap_loop(
   pcap_t *p, // session handle
   int cnt,   // how many packets to sniff
   pcap_handler callback, // callback function
   u_char *user
)

pcap_loop(handle, num_packets, got_packet, NULL);

void got_packet(
   u_char *args, const struct pcap_pkthdr *header, const
   u_char *packet)
```