Exercises
Let’s talk about Distribution and your plans.
Is everyone experienced with make, vi or emacs (or ed) and writing C programs?
Exercises: Build MonP:
A Monitoring Protocol over UDP/IP

- Three Operations supported:
  - Report Packets
  - Report Bytes
  - Print Packet
    » message is supported
    » you will implement the operation

- End systems will generate MonP/UDP/IP messages
- Router Plugins will perform MonP operations
- End systems will display resulting MonP information
- We’ll build up to the final plugin through several steps

- 2 End systems to be shared across all users
- 15 Router systems
Lab Configuration

Server has 15 IP aliases

Server (recv/echo)

Server (recv/echo)

Client (xmit)

Client has 15 IP aliases

Router Plugins (Crossbow)
You will each sit at and work on One of the 15 router machines

Example

Server: cb17
> serv

Client: cb16
> sendpkts

192.168.17.1
192.168.2.1
192.168.2.2
192.168.2.21
128.252.153.216

cb2:
Setup

• Should be no more then 2 people per Router
  – some will have one person per Router
  – If you are a two person group avoid cb12-cb15 (limited space)
• Routers are machines cb1-cb15 (cb13 might be BAD)
• End systems are cb16 (client) and cb17 (server)
• All connected via Ethernet.
• User accounts: ‘cbuser#’, where # matches machine
  – e.g. if you are on cb4, use the account cbuser4
  – passwords: CbTut# (e.g. CbTut4)
• Root access:
  – available to cbuser# on machine cb#
  – password: CbowTut
  – You will probably want to do this after you become root:
    > exec tcsh
Note about Addresses

- The address structures in the Crossbow Kernel are IPv6
  - IPv6 address structures are a superset of IPv4
  - IPv6 address are 128 bits
- We will be using IPv4 addresses
  - e.g. 192.168.5.2
  - IPv4 addresses are 32 bits
- Notation for using IPv4 address in IPv6:
  - Use double colon before address:
    ::192.168.5.2
  - Double colon tells the utilities to set everything to the left to 0’s
Exercises

• Load printPacket plugin and try it out

• Build your own plugin
  – Start with template and add printf(“I got a packet\n”);
  – Use this plugin to try out array of different filters.

• Modify your plugin to print out UDP/IP info
  – Using utility functions used in printPacket
    » print udp header for packets using UDP Port 5555
  – Using filter
    » print udp header for packets using UDP Port 5555

• Implement “Monitor” Protocol
  – Using utility functions used in printPacket
    » test with different command line options to sendpkts
  – Implement Protocol
    » using code module supplied by us
Exercises

• Create two instances of your protocol plugin:
  – use one with a filter to capture packets from client
  – use second with filter to capture packets from server
• Modify protocol plugin by adding PrintPacket command
  – it already exists in sendpktks program
  – plugin code currently just says that command is not supported.
Exercise 0
Exercise 0: Running PrintPacket Plugin

Log in as root and reboot your router machine
When it comes back, log in as cbuser#,
Start X:
> startx

On your machine have four windows:
one as root on cb#
one as user cbuser# on cb#
one as user cbuser# on cb16 (use ssh to connect)
one as user cbuser# on cb17 (use ssh to connect)
Exercise 0: Running PrintPacket Plugin (continued)

on cb#, cbuser# Window:
  cb#> tail -f /var/log/messages
  watch what is displayed here while you do the other steps

on cb#, Root Window: Load Plugin
  cb#> cd /usr/CB/plugins/TUTORIAL/PrintPacket
  cb#> modstat
  [modstat shows you what plugins are loaded. Right now, there should be none]
  cb#> sh loadit 1
  cb#> modstat
  [now you should see your plugin loaded]
Exercise 0: Running PrintPacket Plugin (continued)

cb17 window: Start server program
   cb17> cd /usr/CB/plugins/TUTORIAL/programs
   cb17> ./serv -h
   This will list the command line arguments are
   cb17> ./serv -v -e 1 -l 192.168.<#>.2

cb16 window: Start client program
   cb16> cd /usr/CB/plugins/ TUTORIAL/programs
   cb16> ./sendpkts -h
   This will list the command line arguments.
   cb16> ./sendpkts -v -d 1000 -c pkts -n 1 -l 192.168.<#+15>.2 server<#>
Exercise 0: Running PrintPacket Plugin (continued)

After you have successfully seen the result of the Plugin executing on a packet:

Root Window: Unload plugin
   cb#> cd /usr/CB/plugins/TUTORIAL/PrintPacket
   cb#> sh unloadit 1
   cb#> modstat

   cb#, cbuser# window: Kill your ‘tail -f’ with <ctrl-c>

   cb17# window: Kill your server program with <ctrl-c>
Exercise 1
Exercise 1A: Building your own plugin

On your machine have four windows:

one as root on cb#
one as user cbuser# on cb#
one as user cbuser# on cb16
one as user cbuser# on cb17
Exercise 1A: Building your own plugin

cbuser# on cb# Window:
   cb#> cd /usr/CB/plugins/TUTORIAL/MonP
   cb#> ls
[Nothing there]
   cb#> cp ../TEMPLATE/* .
   cb#> ls
[Now we want to rename the plugin to MonP]
   cb#> mv myplugin.c MonP.c
   cb#> mv myplugin.h MonP.h
[Edit the files: Makefile cbpgi.c MonP.c MonP.h and change all occurrences of myplugin to MonP]
cbuser# on cb# Window:

   cb#> cd /usr/CB/plugins/TUTORIAL/MonP
   edit the file MonP.c and add this to the function my_handle_packet()
       printf("I Got a packet!
");
       printf("#################
");

also, find out what the plugin id code is for your plugin, you’ll need this number later.

   > grep PGICODE *.h

build the plugin:

   cb#> make

   cb#> tail -f /var/log/messages

[You should see the results of your packets being processed by your plugin in this window when you do the next couple of steps…]
Exercise 1A: Building your own plugin

Root on cb#  Window: Load the Plugin ourselves this time!!

   cb#> cd /usr/CB/plugins/TUTORIAL/MonP
   cb#> make load
   cb#> modstat
   cb#> cm

       [ see what the possible cm commands are, and check the args for some: ]
   cb#> cm create_instance
   cb#> cm add_filter
   cb#> cm register_instance
   cb#> cm create_instance <pgicode#>

       [note the instance number that this returns, negative value means error]
   cb#> cm add_filter 1 0 0 0 0  ::192.168.<#>+15>.2/128  ::192.168.<#>.2/128

       [note the filter handle that this returns, negative value means error]
   cb#> cm register_instance  <pgicode#>  <instance#>  <filter#>
Exercise 1A: Testing Your Plugin

**cb17 window:**

```bash
  cb17> cd /usr/CB/plugins/TUTORIAL/programs/
  cb17> ./serv -v -e 1 -l 192.168.<#>.2
```

**cb16 window:**

```bash
  cb16> cd /usr/CB/plugins/TUTORIAL/programs/
  cb16> ./sendpkts -v -d 1000 -c pkts -n 1 -l 192.168.<#+15>.2 server#
```
Exercise 1A: Removing Your Plugin

After you have successfully seen the result of the Plugin executing on a packet:

Root Window:

    cb#> cd /usr/CB/plugins/TUTORIAL/MonP
    cb#> cm deregister_instance <pgicode#> <instance#> <filter#>
    cb#> cm remove_filter 1 <filter#>
    cb#> cm free_instance <pgicode#> <instance#>
    cb#> make unload
    cb#> modstat
Exercise 1B: Various Filters

Repeat Exercise 1A, but try these filters instead.

Try combining different ones. Can you find the most specific filter for the packets you are sending?

You may need to try different command line parameters for serv and sendpkts. Each of them has a “-h” command line argument to print their options

Beware: Very general filters will pick up lots of pkts!!

```plain
cb#> cm add_filter 1 0 0 0 0 0/0 ::192.168.<#>.2/128
cb#> cm add_filter 1 0 0 0 0 ::192.168.<#>.2/128 0/0
cb#> cm add_filter 1 0 0 0 0 0/0 ::192.168.<#+15>.2/128
cb#> cm add_filter 1 0 0 0 ::192.168.<#+15>.2/128 0/0
cb#> cm add_filter 1 0 0 0 ::192.168.0.0/112 0/0
cb#> cm add_filter 1 fxp0 0 0 0 0/0 0/0
cb#> cm add_filter 1 0 udp 0 0 0/0 0/0
cb#> cm add_filter 1 0 tcp 0 0 0/0 0/0
cb#> cm add_filter 1 0 5060 0 0/0 0/0
cb#> cm add_filter 1 0 0 5050 0/0 0/0
```
Exercise 2
Exercise 2A: Modify your plugin to Print UDP Info

Open another window on your cb# machine as user cbuser#.

In this window:

    cb#> cd /usr/CB/plugins/TUTORIAL/PrintPacket

[ take a look at the file PrintPacket.c . We will use some of
the functions that this uses to perform this step in the
exercises. Find where it uses the following functions:

        locateUdpHdrFct()
        printUdpHdrFct()

    cb#> cd ../utils

[ take a look at the file udpHdr.c . Find where it defines the
following functions and where it prints the UDP dst port:

        printUdpHdrFct()
Exercise 2A: Modify your plugin to Print UDP Info

Using the utility functions that are used in the PrintPacket plugin, modify your plugin to print the IP and UDP Headers for packets which are UDP and use UDP destination port of 5555. Add the body of the function filterUdpPacketsFct() to do this.

[Your plugin template already has the include files necessary to use the utility functions. Also, your Makefile is already set up to link the utility functions with your plugin. You just need to add the function calls and any variables you might want.]

Use a very general filter to make your plugin code do the packet selection.
Test your plugin with UDP and TCP packets.
Test your plugin with a variety of UDP port numbers.

Don’t forget to perform the cleanup operations (deregister_instance, …)
Exercise 2B: Modify your plugin to Print UDP Info

Modify your plugin from Exercise 2A to print the IP and UDP Headers for all packets. Comment the call to the function filterUdpPacketsFct() out. We’ll use a filter this time to do the selection. Add the body of printIpUdpHdrsFct() to print those headers. For this we will assume that the packets are UDP. The filter should guarantee this!!

Use a filter that does the packet selection of UDP packets with destination port 5555.
Add ‘-p 5555’ to your server and client command lines
Test your plugin with UDP and TCP packets.
Test your plugin with a variety of UDP port numbers.

Don’t forget to perform the cleanup operations (deregister_instance, …)
Exercise 3
Exercise 3A: Implement FRED Protocol

Using the utility functions that are used in the PrintPacket plugin and code samples from utils/*.c, modify your plugin from Exercise 2B to print **JUST** the packet payload for all packets received.

Choose a filter to use.
Test your plugin with UDP and TCP packets.
Test your plugin with a variety of UDP port numbers.

Look at the packet payload data and see how it corresponds to what we have defined as the Monitor Protocol.
Try different command line options to sendpkts to send different commands. Note the results at the client systems. We’ll want to compare this with the results in the next step...

Don’t forget to perform the cleanup operations (deregister_instance, …)
Exercise 3B: Implement Monitor Protocol

Uncomment the call to the monitor protocol function (ProcesssMonP) and uncomment the function in your plugin code.

Look at the body of the ProcessMonP() function. Most of the code is given to you.

Look for comments of the form:

// BEGIN: ADD CODE HERE to ….    
// END: ADD CODE HERE to ….  

These tell you where you need to add calls to utility functions and do certain tests in the code. You may look at the rest of the code, but don’t worry if you don’t understand it.

Work with the code until you get it to compile.
Exercise 3B: Implement Monitor Protocol (continued)

Choose a filter to use.
Test your plugin with UDP and TCP packets.
Test your plugin with a variety of UDP port numbers.

Try different command line options to sendpkts to send different commands. Note the results at the client systems. Compare these results to those in the previous step…

Can you come up with filters to cause your plugin to operate on packets arriving from just the client?
Just the server? How about both?

Don’t forget to perform the cleanup operations (deregister_instance, …)
Exercise 4
Exercise 4: Multiple Instances

Using your plugin from Exercise 3B:
Create an instance and an appropriate filter to monitor packets arriving from the client.

Create a second instance of the same plugin with an appropriate filter to monitor packets arriving from the server.

Test them and check the results at the end systems.

How do the different instances know whether they are instance 1 or 2?

Don’t forget to perform the cleanup operations (deregister_instance, …)
Exercise 5
Exercise 5: Augment the Monitor Protocol

Using the utility functions to modify the code for the Monitor Protocol to add the PrintPacket command.

Look at the file /usr/CB/plugins/TUTORIAL/fred/fp_proto.c for the code for the actual protocol. You will have to modify it to add the PrintPacket command. When you get it to compile, go back to MonP.

Rebuild your plugin and test this new feature.
END of Exercises