CS 422 (Park) **Assignment IV** **Due: Mar. 28 (Mon.), 2011**

Submission instructions: Please type your answers and submit electronic copies using `turnin` *by 4pm on the due date. You may use any number of word processing software (e.g., Framemaker, Word, LaTeX), but the final output must be in pdf format that uses standard fonts (a practical test is to check if the pdf file prints on a CS Department printer). For experiments and programming assignments that involve output to terminal, please use* `script` *to record the output and submit the output file. Use* `gnuplot` *to plot graphs.*

PROBLEM 1

For students who have the 5th edition (Comer): Read chapters 16 and 17. For students who have the 4th edition: Read chapters 11 and 12.

PROBLEM 2 (50 + 30 pts)

(a) The purpose of this problem is to introduce sniffing raw data from the wire—more precisely, Ethernet interfaces of sender/receiver PCs—on the machines in **LWSN B158**. **For hardware set-up reasons, please use the machines in LWSN B158 for Problem 2 only**. The `sslab` machines have been set up such that pairs of machines 01–02, 03–04, ..., 23–24 are point-to-point connected using dedicated NIC cards separate from ones that provide general switched network connectivity. Execute the following command on sender and receiver PCs to capture Ethernet frames:

```
% sudo /usr/local/etc/tcpdumpwrap-eth1 -c24 -wEX
```

The command will capture 24 packets from the point-to-point Ethernet LAN and save them in the file `/var/tmp/`*login*`-EX`, where *login* is your login ID. Generate your own traffic on the private network by doing `ping` from sender machine to receiver machine (`ping -c count -s 10 receiver-addr`) where `count` is the number of packets transmitted.

Since each PC is dual-homed, to distinguish the dedicated NIC card for the point-to-point link from the general-purpose NIC card, the symbolic name "here" has been set up to refer to the IP address of a PC's dedicated NIC card, and similarly for "there" which translates to the IP address of the receiver PC's dedicated NIC card. For example, at host `sslab01`, executing `ping there` will generate ping packets going out on its second Ethernet NIC (`eth1`)—which has IP address 172.16.25.101—that arrive on `sslab02`'s secondary NIC card which has IP address 172.16.25.102. Choose a value of `count` so that `tcpdumpwrap` captures at least 24 packets. Submit the log file in hexadecimal format. Check if the log files generated at sender and receiver PCs are identical. Save your `ping` terminal interaction using `script` and submit it along with the hexadecimal `tcpdumpwrap` log file.

Using the Ethernet header format discussed in class, decode from the hexadecimal dump what the values for the fields in the Ethernet header are. Do it manually (i.e., visual inspection), not by automated tools. Use `ifconfig` to compare the Ethernet addresses that you have captured with those given by `ifconfig`. From the value of the type/length field determine whether the Ethernet frames are DIX or original IEEE 802.3. Explain how you are able to distinguish between the two and what the specific value found indicates. Determine if any padding is being done in the Ethernet frame to satisfy its minimum payload requirement. In the case of DIX frames where there is no length field, how can the payload be distinguished from padding? In Ethernet's payload, locate the 20 byte IP header (`ping` implements a protocol called ICMP which uses IP) and find the values for the source and destination IP addresses. Compare the values against those obtained by `ifconfig`. Locate the version field of the IP header and check its value.

(b) An anonymous `ftp` server (i.e., daemon) has been set up that listens to client requests arriving on the dedicated secondary Ethernet interface of a PC in LWSN B158. Execute `ftp there`. When prompted for user name enter "anonymous"; when prompted for password press the `return` key. First, execute `ls` to check the content of the server's current directory. Second, use `get` to fetch the file `CONTENTS`. Before executing `ftp`, run `tcpdumpwrap` at the client so that all packets exchanged in the `ftp` session over the dedicated point-to-point link are captured. Submit the log file in a format of your choice. Inspect the captured packets and their payload, and try to make sense of what you find. `ftp` runs on top of TCP which, in turn, runs on top of IP. This implies that the payload of Ethernet is IP (as in the `ping` example) whose payload is TCP (header plus TCP's own payload). Can you extract the user

name or password from the sniffed data? What about the contents of the subsequent client/server interaction (`ls` and `get`)?

PROBLEM 3 (60 pts)

Rewrite the remote command client/server application of Problem 5 (use the improved server code of Problem 6), Assignment II, such that the client and server use TCP sockets (in place of FIFO) to communicate over a network. From a programming perspective, the internals of how TCP (or IP) works is not needed to use it. Like FIFO, a socket is one of the 7 file types in UNIX-like operating systems (including Linux) that is created by a `socket`() system call that returns a file descriptor. TCP is selected by using the SOCK_STREAM option. 4-byte IP addresses are used to identify a destination machine (more precisely, a network interface on the machine if it's multihomed), and 2-byte port numbers are used to identify the process that data transferred by TCP is to be delivered. The details of necessary system calls, default options, and header files will be discussed during the PSOs. Benchmark the TCP client/server application as before, recording the terminal interaction using `script`.

From a DoS vulnerability viewpoint, can you think of additional issues that may arise in the networked remote command client/server application that did not exist in the single host case? What about the opposite scenario—vulnerabilities that exist in the single host case that do not apply in the networked case?

PROBLEM 4 (70 pts)

Modify the client/server file transfer application of Problem 6, Assignment III, such that UDP (SOCK_DGRAM option in `socket`()) is used in place of TCP. Unlike TCP, UDP does not retransmit lost packets, its main function being adding port numbers and IP addresses to payload. Although UDP can be used with `write`() and `read`() system calls, use the UDP-specific system calls `sendto`() and `recvfrom`() in your client/server code. At the client, implement an additional coding change such that instead of making a blocking `read`() (in our case `recvfrom`()) system call awaiting the next packet from the server, the client instead registers a signal handler `my_receive_server_data`() with the kernel such that the handler is invoked when SIGPOLL (or SIGIO) is raised which happens when a packet arrives at the kernel. Thus the client sleeps until it is woken up by a packet arrival interrupt, processes the packet and sleeps again. Although not very likely (due to, on average, low traffic), UDP packets may go missing in the lab's Ethernet LAN, for example, due to buffer overflow. Perform the same benchmarks as before and submit the resultant output. Compare the received file against the original and check if file transfer was error-free. Note that file transfer using UDP (e.g., `tftp`), despite its unreliability, is sometimes used in practice in place of TCP when a network does not frequently drop packets and speed is at a premium (UDP has minimal overhead compared to TCP which implements sliding window ARQ and congestion control). Compare the performance results when transporting files over an Ethernet LAN using UDP against the FIFO case of Problem 6, Assignment III. Give an assessment of what you find.