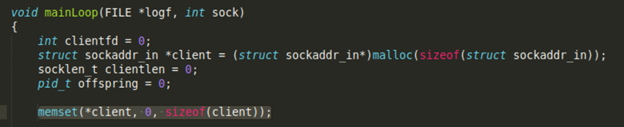
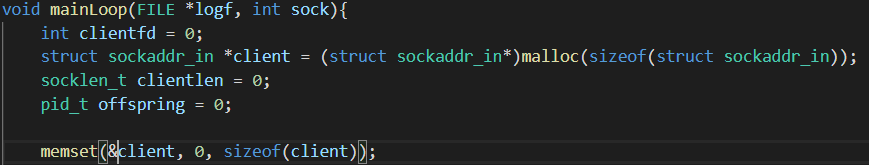
CYSE 411-002

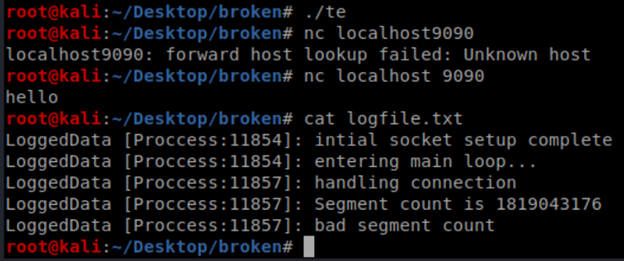
Code Analysis Phase 1

Robert Weiner, James Cogswell

**Getting it to compile:** To get the code to compile all we had to do was change the memset() parameter in mainLoop() from “\*client”1 to “&client”2. Before, the code was trying to send the value at the pointer “client” into memset(), and after we are sending the address of the client struct into memset. As memset() is expecting a pointer, this is required. A screenshot is provided showing that logFile.txt is generated upon connection to the socket3.

1

2

3

**What the code does:**

* Basic Purpose: Creates an IPv4 TCP socket on port 9090 which is controlled by a child process running in the background which hosts a remote shell. This shell allows for authentication and authorization of users and admins. As a user you can create and view written text files without directly accessing the filesystem of the server. As an administrator, this program allows for the functionality of creating new users and issuing system level command line commands remotely. The remote shell can be accessed locally with “nc localhost 9090”.
* Inputs: All user inputs are handled by readSock()
  + readSock() reads in user input using the standard recv() function until the length fed in is reached. The read in data is fed into the buffer that was fed into the function. The categories of inputs are the user’s logon info, commands to the shell, and appropriate input for the commands initiated (such as a system call string, an article to be written, or a new user’s username and password).
  + readSock() is called from:
    - writeArticle() to receive the user’s input to be fed into the new article. This is covered more in depth in “outputs>File System>Writing an Article”
    - adminFunctions() for receiving the admin’s action (see last bullet of this section for how the data in parsed)
    - userFunctions() for receiving the user’s action (see last bullet of this section for how the data in parsed)
    - handleConnection() for receiving and parsing out the username and password to feed into authenticate(). The username and password are parsed from the raw input by calling findarg() with the argument type (username or password) being looked for
    - How the action is parsed: The first character of the action string is compared to a set of defined characters that mean certain action types (“ is list articles, # is read article, $ is write article, % is command, and & is new user). The appropriate function is called after this comparison. The action associated with this action type is the rest of the user entered string after the first character. This is accessed in the functions with “&action[1]” which is a pointer to the second character is the user input array. Basically this is a string containing everything but the first character the user entered.
* Outputs: The outputs of this program go to three general locations-
  + What the user sees / STDOUT:
    - All STDOUT data is controlled by writeSock(). This function uses the standard send() function to send the data fed into the function to the user on the other end of the socket’s STDOUT. This is the only function that sends outbound data to the user.
    - writeSock() is called from
      * listArticles() to list all articles they have written
      * readArticle() to display an already written article of the user’s choosing (or display the article doesn’t exist)
      * adminFunctions()/userFunctions() to show the shell is ready for a new command
      * handleConnection() to display if the username or password was incorrect
      * writeArticle() to say the file doesn’t exist
  + logfile.txt:
    - All logged data from the execution of the program is sent to logfile.txt using logData(). This function accepts two or more parameters: the file pointer for the log file, the base string which may contain formatting expressions, and a possibility of more parameters which are the arguments to be inserted into the format string. These arguments are accessed with the va\_start() function and inserted into the format string using vsnprintf(). Once this is done, the newly formatted string is written to the log file with fprintf(). When being written by fprintf(), the string is inserted into a new string containing the pid of the process. After writing to file, fflush() is called to flush the kernel file write buffer.
  + File system: Output to the file system is in two ways: Creating a new user and writing an article-
    - Creating a new user: Adding a user is done entirely by the addUser() function which is called within adminFunctions(). When calling this function, the action string will be in the format “&<username>:<password>”. This function parses the action string to extract the username and password by locating the index of the colon, saving this index+string pointer to a new pointer pointing to the start of the password and then creating a new user file through a system echo call (never actually called) using a formatted string built in snprintf().
    - Writing an article: Writing an article is done entirely by the writeArticle() function. This function creates a string holding the path to the new article to be written (by concatenating the file name in the action string being fed in), then opens the file for writing. It then makes calls to readSock() and uses fputs() to write this user entry to the article file. It continues calling readSock() until the user’s input contains the substring “!!!”. At this point, it null terminates the user input and writes the last input to the article file and exits.

Functional Diagram: If this image is hard to read, please navigate to [draw.io](http://draw.io) and open the file attached to the submission to view our functional diagram in full quality

