CMPE 156/L, Spring 2014
Final Project
Phase 1 Due: Tuesday, May 27
Phase 2 Due: Tuesday, June 10

The goal of this project is to develop a proxy HTTP server that can process HTTP requests generated by a Web browser and perform two basic functions:

1. Filter requests based on an access control list.
2. When the destination of the request supports Secure HTTP, translate the unsecure requests generated by the browser to use Secure HTTP using the OpenSSL library.

The project will be done in two phases. In Phase 1, you will develop the filtering function, and will add the security features in Phase 2.

General Features

- The proxy server needs to support HTTP 1.0 and 1.1. It needs to handle only the GET, HEAD and POST requests.
- A standard browser client (e.g., Firefox) will be used to generate the HTTP requests. You need to configure the browser to direct its requests to the proxy server by setting up the browser with the IP address and listening port number associated with the proxy server.
- Only one copy of the browser will be run at a time, but the server must be designed to support concurrent requests from the browser.

Phase 1 Requirements

In Phase 1, the proxy server needs to have the capability to parse the headers of incoming HTTP requests and forward them to their ultimate destinations if allowed by the access control mechanism. It should then forward any data received from the destination back to the browser.

The list of permitted sites is maintained in a file *permitted-sites.* An example looks like:

```
www.ucsc.edu
www.soe.ucsc.edu
www.wikipedia.com
```

The proxy server must return an HTTP error response 403 (Forbidden URL) to the browser if the access is to a site that is not in the *permitted-sites* file. If the proxy server received an HTTP request method other than GET, HEAD or POST, it should return an error code of 405 (Method not allowed) or 501 (Not implemented). The proxy server must also respond to any errors in the request (for example HTTP header fields missing or inconsistent) with the appropriate responses (for example, 400 Bad Request).

The proxy server must always remain open and accept requests from the browser.

The proxy server does not need to support *https* in Phase 1. It also does not need to support the chunked transfer mode of HTTP 1.1.

Logging: The proxy server should maintain a log file to track all requests received. It should print one line for each request with the following information:

- Date and time the request was received
- Type of request and HTTP version
- Requesting host name
- URI and server address
- Action taken by proxy (forwarded, filtered, etc.)
- Type of errors detected, if any

Persistent connections: Note that the browser will use persistent connections by default if it is using HTTP 1.1. In this case, either the client (browser) or the server may initiate the close. The proxy server must be able to deal with this by closing the connection on the other side when the TCP connection on one side closes.

Phase 2 Requirements

In Phase 2, you will add security features to the server side of your proxy server using the OpenSSL library. You can assume that the browser will always use unsecure connections to the proxy server. When the server supports *https*, the proxy server must first authenticate the server and subsequently transfer all the session data over an encrypted connection. When the server sends data to the client (browser), the proxy server must decrypt the data and forward it to the client as cleartext. You will use the OpenSSL library functions to perform the authentication and encryption/decryption tasks.

What to submit?

You must submit all the files in a single compressed tar file (with `tar.gz` extension). The files should include

1. A design document summarizing the internal architecture of the proxy server, with details on the following questions:

    a. How does the proxy server manage its TCP connections on the server side and the browser side. When does it open and close connections?
    b. How do you support concurrency on the server side?
    c. What are the steps involved in processing a request from the browser side?
    d. How does the design deal with errors?
    e. (Phase 2) Decribe your OpenSSL implementation.

2. All source code necessary to build and run the client and server

    a. Optional: Any test code you used to test your design.

b. A Makefile that can be used to build the client and server binaries.
c. A README file including your name and a list of files in the submission with a brief description of each. If your code does not work completely, explain what works and what doesn't or has not been tested.

Grading

Each submission will be tested to make sure it works properly and can deal with errors. Grades are allocated using the following guidelines:

Basic Functionality of code:	40%
Dealing with errors	20%
Design and documentation	25%
Style/Code structure, etc.	15%

Each phase requires a separate submission through eCommons, but it is assumed that the submission in Phase 2 will be enhanced versions of the documentation and code in Phase 1.

Honor Code

This is an individual project and all the code must be developed independently. It is not acceptable to submit code from the Web (or anywhere else). It is acceptable to look at any code and borrow ideas, but the code you submit must be written by you. If you borrow specific ideas from code not written by you, you must acknowledge this is your source code and in your README file.