cs3157 – Advanced Programming
Summer 2006, lab #2, 20 points
May 31, 2006

A lab in CGI using Perl...

Follow these step-by-step instructions. This lab must be submitted electronically (see instructions at the end of the lab) by Monday June 5, 5pm.

You need to include a README file with information about each file submitted and any other comments you want the TA to see. Reminder, please ask for help before getting stuck….also please complete the last step ASAP as I would like everyone to submit solutions before we discuss it in class next week. Do not worry if you need extra time, but please start early and contact me about problems you might be facing. Also please work in an IDE, contact me if you need more advice, I will demo eclipse at the beginning of next class.

Remember that
```
use strict;
```
is required for these assignment. If there is something wrong with your code, you will see an error in your browser when you try to call the script. One way to test, is to actually run the script from the command line to see if the interpreter complains about anything. Also, although everything can go into one he "main" its better to divide your code into a series of subroutines, both for coding and debugging purposes.

Feel free to refer to class notes on CGI. While it is your responsibility to understand everything in the lab/course at the same time please feel free to ask as many questions as you want. Remember to read through each step entirely before starting and to have fun!! If something is not specified, either ask or use your best judgment, and document in the README your decision.

Step 1) Setting up your environment, and using HTML: (3 point)

You will need to setup your local machine to run CGI, as discussed in class. Please download and install Abyss Webserver (as shown in class). Remember our discussion on security issues. I would recommend setting it up for manual launch (but keep that in mind when testing).

In the installed directory (c:\Program Files\Abyss Web Server\) there is a directory called htdocs, which will host the top level directory for your files. That is when you go to http://localhost this is the starting directory.

Create a lab2 directory to hold your work. Now create a simple html file called **step1.html** in the lab2 directory. This page should contain the following elements.

1. your name, with last name in bold
2. A page title in the html head section.
3. A small saying (anything you like more than a few words).
4. An image (copy the image so it sits in the same directory).
5. A link to the class webpage, to open a new webpage (Hint: lookup target)

Feel free to make the webpage as fancy as you like.
Check that the page has permission to be viewed by typing http://localhost/lab2/step1.html in your browser window.

Step 2) Learning the difference between GET and POST (6 points)

As shown in class and on the notes, you will need to setup the perl environment to work correctly. In this step, you will create an html page and 2 separate cgi scripts. The HTML page (**step2.html**) should have the following code in it:

```
<html><head><title>Step2 webpage</title></head>
<body>
<form action="step2a.pl.cgi" method=POST>
  ?????
<input type="submit" value="test post"></form>
<form action="step2b.pl.cgi" method=GET>
  ?????
<input type="submit" value="test get"></form>
</body></html>
```

First replace ????? with a text box input type and one other input type of your choice.

In both **step2a.pl.cgi** and **step2b.pl.cgi** you need to process and display the input passed in, as a simple html page saying

```
I was given Key = …. With a value of = …
```

(for both input types).

Hint: This means you need to parse out the arguments passed in. remember to first split around the & sign and then around the equal sign. Example:

```
my ($pairs,$key,value);
foreach $pairs (split /&/ $inputstring){
#now cut the $pairs around the equal sign …..
```

Hint2: one of the methods passes user input through ENV and one passes it through STDIN, but you still ENV for the length (see read method in perl)

HINT3: the scripts are basically the same, get one to work and then copy and convert it for the other task.

Question: can you figure out the difference between when you would like to use POST over GET??

Step 3) Login system (8 points)

We will be creating a login program to authenticate visitors to your website. The CGI script should be named **step3.pl.cgi**.

Since you don't want private information on the URL, the script expects everything over POST.

If nothing is passed to the script, create an html page which will show the following elements:

1) Some text on top (ACME incorporated or be creative)
2) a form which uses the POST method
3) It should contain 2 text fields, one which asks for a name, and one which asks for a password but shows stars when typing input. (we discussed in class how to make it print stars when you type in a password)
4) end

Else your CGI script (**step3.pl.cgi**) should work in the following manner:

1) It should read a file called **passwords.txt** (you need to create it manually ahead of time with at least 2 usernames and passwords).
    a. Feel free to reuse last week's assignment
    b. Need to scramble the passwords using md5 in the txt file
    c. passwords.txt contains on each line a user name and password separated by an colon (:) sign, with the password md5 encoded.

2) Read in the username and password passed in from the html page.
3) If you match a username but not the password, print out an html page with a bad password entered message.
4) If you don't match a username print out an html page with bad login message.
5) If you match both, print out a success log in message.

Technical Coding Specifications:

You need to define the following subroutines and use them in your code:

1) ($username,$password) = **getUserPassfromWebCGI**(\%ENV);
    a. That is the call returns a pair of user name and passwords which we read in from the web.
    b. Hint: return ($a,$b) will create an anonymous array and return a pair value.
2) **readUserPassFile**($filename,\%usernamepass)
3) **matchUserPassword**($username, $password, \%usernamepass)
    a. return true if we match both user and password in the hash named usernamepass
4) **isUserOK**($username, $password, \%usernamepass)
    a. return true if username is in the usernamepass hashmap

Hint: all arguments to a subroutine are passed in the @_ array, so that the first element is in $_[0] etc
HINT2: when you pass a reference to a hash, you derefernec it as follows: `$$hashref{$key} = $val;`

Step 4) Serving Images (2 point)

For this assignment you need to write a perl script to serve images. This is a very simple task. Instead of printing out text, you will print out the contents of an image file (you can use the image used in step1). Name the script **step4.pl.cgi**.
When loaded, the script simply loads an image to the browser.

Reminder hint: BINMODE

Step 5) manipulating telnet session (4 points)

We played around with telnet at the end of class. Write a **server.pl** and **client.pl** program to allow a server to listen for connections and a client to connect for sending information. We want to have some fun, so what ever you type on the client side, is printed on the server side, and is also sent back to the client, but backwards and then printed out by the client ☺

NOTE: use IO::Socket::INET;
to get the socket library. Choose a random port to listen on (higher than 1500) and start you server to listen for a connection (your script should take a port number as the first argument (hence the $portnum scalar we saw in the notes). It should out put anything it reads and prepend "server:" before printing out the stuff, so we can tell it was printed from the server script.

The client should also take port number, and host machine (where the server is running) as command line arguments, and allow it to redirect your local STDIN and STDOUT from the server. So for example if you start in one window,

```
./server.pl 9000
```

and in the next window,

```
./client.pl 9000 localhost
```

it should allow the client STDIN to be sent to the server, the server takes whatever the client types, and sends it back (but backwards) and it also prints out whatever was sent on its side, with the words server first.

Hint: see cpan module docs

Hint: my $server = IO::Socket::INET->new(LocalPort=> $portnum,
LocalAddr => 'localhost',
Proto => 'tcp',
Reuse => "1",
Listen => "10") or die "could not start server on port $portnum ….\

Submit your lab.

You need to submit a few files for this lab: **README, step1.html, step2.html, step2a.pl.cgi, step2b.pl.cgi, step3.pl.cgi, step4.pl.cgi.**

See the class page for submission instructions.