NS Lab 2A – HTTP and SMTP

Hao Zhu (h.zhu@uva.nl)
Rex Valkering (rexvalkering@gmail.com)
Sam Ansmink (sam.ansmink@student.uva.nl)
Koen Koning (koenkoning@gmail.com)

Hand-in time (submit to blackboard) by November 13 11:59CET
Total points: 15

Abstract
This assignment focuses on learning the basics of TCP socket programming and HTTP and SMTP protocols.

This lab must be done individually.

Preparation

For this assignment you must use Python 2.x. It is already installed on the lab computers; otherwise you can download it from http://www.python.org. If you do not know Python, learn it. It is a very simple language.

You also need to read the documentation on the socket module that is included with the standard Python distribution. Python's socket module is a direct translation of the Berkeley Sockets API for C. You can find examples on socket programming in Python in your textbook, section 2.6 or you can Google it.

Socket module documentation: http://docs.python.org/library/socket.html

Submission

Submit your working code in an archive called lab2-<yourname>.zip (or 7z, or tar.gz, etc). It should contain the following files:

- lab2a-<yourname>.py (task 1 + 2)
- cgi-bin/<yourname>/email.py (task 3)

Where <yourname> is <last name plus first letter of first name>, for example lab2a-koningk.

You must also write your full name and student number at the top of the files (in comments).

Task 1 – Basic HTTP Server (6 pts)

The first task is to implement a simple web server that can handle a single client at a time (in lab 2B you will learn how to handle multiple clients). You must parse HTTP requests sent by the client (web browser) and send appropriate responses. You may ignore any request headers for now.

If the client sends a GET request respond with 200 OK and the file contents. If the file does not exist respond with 404 Not Found.

If the client sends any other kind of request respond with 501 Not Implemented.

After the response is sent close the connection and wait for the next client.

HTTP responses normally contain a number of response headers in order to control the transaction. Implement (at least) the following response headers: Connection, Content-Type, and Content-Length.

Take care to follow the protocol exactly or the HTTP transaction will not complete and you will lose points. Use the web console inside your favorite browser to check that your server works correctly. You can test by serving your own website.

Although Python includes an HTTP server module, you are not allowed to use it for obvious reasons.

The HTTP protocol is described in RFC 2616: http://tools.ietf.org/html/rfc2616

Task 2 – Common Gateway Interface (4 pts)

CGI is a standard interface for web servers to serve dynamic content rather than static pages. It works by executing scripts as external processes and redirecting stdout as a response back to the client. The parameters are given by a number of environment variables that the script can read and process. It is usually used to execute Perl scripts, but in this case you will use it to execute Python scripts.

Your task is implement a subset of CGI, as follows. Whenever the client requests a URI starting with /cgi-bin, you must execute the requested Python script from the *cgibin* directory instead of sending its contents directly. The HTTP server must output the HTTP status line, but the headers and contents of the response are output separately by the script. The script outputs by printing to stdout, so the server must pipe the script process's stdout to the client socket.

You must pass (at least) the following environment variables to the script:

- DOCUMENT_ROOT: The public_html directory.
- REQUEST_METHOD: The HTTP request method that was used to access the script.
- REQUEST_URI: The requested URI, without query string.

- QUERY_STRING: The query string following the URI, if any. Example: In /cgi-bin/hello.py?name=pietje the part before the ? is the URI, and the part after is the query string.
- PATH: The standard PATH environment variable copied from the HTTP server's own environment.
- There are many more CGI variables, but you do not have to implement them.

For testing, write a script accessed from /cgi-bin/env.py that outputs all environment variables passed to it.

Tip: Use the **subprocess** module to execute the Python CGI scripts.

CGI is described in RFC 3875: http://tools.ietf.org/html/rfc3875

Task 3 – Authenticated SMTP (5 pts)

In this task you must use the CGI functionality from the previous task to write a script to send an email with parameters given via a web form that is filled in by the user.

The web form is located in the file public_html/email.html. When the user presses the Send button, the browser sends a GET request for /cgi-bin/email.py with the fields and their values in the query string. (Side note: Normally this is done with a POST request with the query string in the request contents, but the distinction is not important for this exercise.)

The task is to write a CGI script to send emails using the SMTP protocol including authentication. Use the STARTTLS and AUTH commands to send emails over an encrypted and authenticated channel. STARTTLS and AUTH are described in RFCs 3207, 4954 and 4616.

Use **smtp.uva.nl** as a test server. If you work from home, note that some ISPs block port 25. If yours does you can try port 587 instead.

Use the *username* and *password* fields in the email web form for the authentication.

At every step in the protocol you must print the server's responses, check the response codes and quit in case of an error.

Although Python includes an SMTP module, you are not allowed to use it for obvious reasons.

Tip: Implement TLS using the *wrap_socket* function inside the **ssl** module.

References

SMTP: http://tools.ietf.org/html/rfc5321
STARTTLS: http://tools.ietf.org/html/rfc3207
AUTH PLAIN: http://tools.ietf.org/html/rfc4616
SASL: http://tools.ietf.org/html/rfc4954