Università degli Studi di Trento

Web Architectures

Extending the simple web server

Gianvito Taneburgo

September 22, 2016

1 Introduction

The following report describes the procedure that has been followed to develop a simple web application that allows users to remotely execute processes on a machine.
The solution employs a client-server architecture, in which HTTP packages with requests and responses are exchanged according to a protocol via sockets.
A process continuously running on the server waits for client requests on a specific port and, as soon as a request arrives, a new thread is started to handle it.
Clients can send HTTP GET packages specifying the command that will be executed on the server. The format of the HTTP package header must be of this kind:

```
GET http://host:port/process/cmd?op1=param1&op2=param2 HTTP/v
```

where:

- *host* is the IP of the server;
- *port* is the port of the server process;
- *cmd* is the command that will be executed on the server;
- *opN=paramN* is a list of key-value pairs, separated by the *&* symbol. *paramN* is the N-th argument that will be passed to the command. Arguments are optional and are provided, if desired, after the *?* symbol.
- *HTTP/v* is the HTTP protocol version used to exchange data (it follows the RFC specification [1]).

[1] https://tools.ietf.org/html/rfc2616#section-5.1.

2 Implementation

The server side of the application is made up of a Java program. Since the clients and the server communicate via sockets, newer versions of Java have been preferred to the elder ones due to the support of the *try-with-resources* statement. This feature is handy when dealing with sockets' input/output streams and enables to write a much more readable and concise code. The application is thus developed using Java 8 and is composed of two classes: *TinyHttpd* and *TinyHttpdConnection*.
TinyHttpd main method is responsible of binding a *ServerSocket* on a user-defined port (default is 8000) on which clients connection requests arrives. A *TinyHttpdConnection* object is instantiated for each of them and a new thread with low priority is started.
The request handling procedure is straightforward. The HTTP package header is first checked with a regular expression that verifies the compliance to both the RFC and the application specifications. If an error is present an HTML page is returned to the user, otherwise the command and its eventual parameters are extracted from the URL. The external process is then started on the server by using the *Process* class and the *TinyHttpdConnection* thread waits for its completion, either successful or unsuccessful. Finally, both the output and the error stream of the external process are entirely read to get the execution information that will be returned to the user in an HTML page.

3 Deployment

Download the *src.zip* file with the source code of the server and open a terminal in the download location folder. To unzip and compile the program just type:

```
$ unzip src.zip; javac src/tinyhttpd/TinyHttpd*
```

To run the server on the default port (8000) just type:

```
$ java -classpath src/ tinyhttpd.TinyHttpd
```

The server is now running and can accept new connections. To list the content of the folder where the server is running, for example, visit http://localhost:8000/process/ls in a browser or directly send an HTTP package by typing in a new terminal:

```
$ curl -H GET http://localhost:8000/process/ls
```

Similarly, to list the processes running visit http://localhost:8000/process/ps?op1=-aux or directly send an HTTP package by typing in a new terminal:

```
$ curl -H GET http://localhost:8000/process/ps?op1=-aux
```

4 Comments and notes

For obvious security reasons the web server should only be used for academic purpose and not made publicly reachable: a malevolent user, for instance, would be able to run arbitrary commands on the machine. The current version of the server provides no sandbox-like mechanism to run processes.