





* All connections as specified are Secured with TLSv1.2, with 256-bit keys generated with RSA.
  + (\*ideally TLSv1.3 but python2 does not support 1.3 without backporting)

**Project Requirements**

The **constraints** of the project require that:

1. the *Testing-Server* and *Question-Server* must execute on different computers (hardware), and must not assume access to any shared (networked) files.
2. the *Testing-Server* and *Question-Server* must be written in different programming languages (selected from Java, C, C++, or Python).
3. the *Question-Server* must generate and assess questions written in Java, C, C++, or Python (only one needs to be supported).
4. your project may be developed for Linux, Windows, or macOS (or a combination).
5. your software must support two or more students simultaneously attempting questions.
6. all network traffic must be encrypted using SSL.
7. your *Testing-Server* does not need to produce a fancy web-interface - basic HTML involving forms, a *textarea*, and *radio*-buttons will be sufficient.
8. the *Testing-Server* does not need to support full user or password management - just logging in and out. Assume that all (named) students and their unchangeable passwords are pre-recorded (there is no need to use UWA student numbers or Pheme authentication).
9. neither the *Testing-Server* nor *Question-Server* require a sophisticated database to manage the students, marks, and questions - simple text files will be sufficient.
10. you should employ the core networking functions (classes, methods, libraries,...) of your chosen programming languages and not employ specific 3rd-party frameworks or resources to complete large parts of the project. If in doubt, please ask.

Works as per project guidelines, except that there is no defined “log out” activity. Simply closing the web browser is “logging out”, and requires a login again, due to the nature of how sessions are managed in the testing server and question server.

Assumed that quiz serves a class of say, 200 students, nature is very “bursty”, i.e. a class of 30 people might take the quiz at once from a single lab. Furthermore, could handle MULTIPLE TestingServers connecting to our QuestionServer, and users are able to login from different browsers/computers and still maintain the same quiz state.

Used Java for Testing Server (version 8) and Python (version 2.6) for project:

* Initially planned to use Java and C, but C implementation became very messy and hard to work with. Ended up going with Python as I wanted to switch to OOP approach that would be easier to understand and work with.
* Figured project does involve a bit of string processing, which python and java can handle fairly well.
* Python also doesn’t need compilation – bonus – less technical issues
* Downside of python – Doesn’t really have “proper” encapsulation… OOP is kind of “bolted on” but encapsulation isn’t really useful for security anyway.
* If I had the option I’d just use Java for everything.

# **Testing Server**

Written in Java 8

HTTPS Server – Quiz with Username/Password Authentication

* Passwords are not stored, cryptographic SHA-1 (salted) used to store password information.
  + *Ideally should be SHA-2 (512 bits) or even SHA-3 – should’ve been set*

Serves simple static HTML pages. Has several “template” html files stored, information is subsituted in and then sent as response to HTTP POST queries.

Based on HTTP POST query, particular web pages are sent in response.

Invalid POST queries and GET queries are merely ignored – No errors sent/printed.

* + Might have been a good idea to have 404 message for invalid pages.

Has a rudimentary leaky bucket implementation to implement rate limiting (i.e. one host cannot flood the program with too many requests), in practice not really a defence against DdoS attacks (which may be provided by some external service i.e. cloudflare) but still useful for my own understanding.

**Startup**

On startup, first reads cmd line arguments for possible parameters – i.e. use SSL/TLS to secure connections, what HTTP port, what port to use to connect to Q server, etc.

Then sets truststore location (for connection as client to Q Server), and attempts to contact Question Server. If TLS is being used, will conduct the TLS Handshake (authentication/connection) and open a socket descriptor that is secured with TLS. If successful, this creates a QuestionClient object which is an attribute of the TestingServer object.

\*Note: We are using self-signed certificates for this project, in reality these would be signed by root authority.

Passwords and template HTML pages are loaded into the program into their arrays (class variables of Thandler/TestingServer object.)

Finally, *HttpsServer* Object created and initialised, (class which wraps HTTP request headers for us), which serves our website.

**How Logins/Sessions are managed**

Uses HTML Hidden Field method.

https://www.journaldev.com/1907/java-session-management-servlet-httpsession-url-rewriting

* Several methods
  + Pass authentication credentials from login. Doesn’t save if user quits then logs in again.
  + ***HTML Hidden Field –*** Create session ID upon successful login. Pass session ID as a html hidden field on every webpage.
  + \*URL rewriting – perhaps a better approach, appends the session ID when receiving the HTTP request.
  + Cookies – session ID in cookie, which is added to request header.

Decided to go with HTML Hidden Field approach – Works with browsers even without cookies, allows seamless login from different browsers, sessions are managed server side,

* + Introduces an exploit where if a user opens developer console and gives away his session ID, another user can access the session. (F12)
  + In reality, a session managing API would be used and dynamically switch between these methods (cookies and URL/HTML hidden field).

**How HTTP POST requests are processed.**

Case A: Authenticate Login, Generate Session  
Case B: Already Logged in…

1: Generate Quiz Questions  
 2: Select Next Question  
 3: Select Previous Question  
 4: Submit this question for marking

5: Invalid. Send bad session error message.

Case C: Invalid Request – Return to Login (home) page.

\*Perhaps should’ve had a 404 in here

**What happens on Login**

Login data is sent as HTTP POST request. (username, password, submit) Follows Case A.

Testing Server authenticates the password with its PasswordAuthenticator Object. This creates a cryptographic hash of the password and username, and compares this with stored data in store/passwords.txt.

**If both fields match,** create a new random ID and make a new SessionHandler, and append it to the TestingServer/Thandler’s session list.

[https://en.wikipedia.org/wiki/Universally\_unique\_identifier#Collisions](https://en.wikipedia.org/wiki/Universally_unique_identifier" \l "Collisions)

\*Possibility of collisions is considered so low that collisions are not detected. If a collision were to occur, then the users who share the same session ID would all see the session of the first user to be assigned this ID.

The SessionHandler holds the question data for the quiz, the state of the quiz (whether it has been generated, which questions have been answered), as well as inheriting the QuestionClient object first generated by the TestingServer (to be used when generating quiz questions and querying during question attempts).

Exploits/Failures:

Currently does not limit number of attempts before entering cool down period – Initially this was supposed to be implemented with a leaky buckets implementation however was never completed due to time constraints.

**How Sessions are Managed after Login**

Session Ids are passed through HTTP POST requests. In the Thandler’s handle\_post() method, the state of the current webpage is determined. At every state except the login state (case B), the session ID passed in the POST request is checked against the list of sessions. If this session ID is found in the list, the SessionHandler object it corresponds to is passed on to the appropriate method.

**Generate Quiz Page**

Until this step, the quiz questions have not yet been generated. Clicking the button to generate the quiz sends the appropriate post request to the testing server, which then (through the QuestionClient object) queries the Q Server for the Question data. When the question data is received from the Q Server, it populates the SessionHolder object’s Questions array. As a response, the Testing Server serves the question.html template, filled in with data from the first question. Sends INIT\_SESS to Q Server, then sends 10 GET\_Q requests in a loop for every question.

Exploits/Failures:

Renaming the generate field (dev console) to “prev” or “next” will cause a socket connection failure error to display in browser.

**There exists a “bug”** with implementing secure sockets between java and python. <https://stackoverflow.com/questions/11787467/java-server-ssl-socket-and-python-client-ssl-socket-problems-when-server-send>

<http://www.stuartcheshire.org/papers/NagleDelayedAck/>

Combination of Naggle algorithm on the server side, and delayed ACK in client’s TCP stack. There is a slight delay between packets. Read() pulls all the bits that HAVE arrived, but not all the bits have arrived yet. (although this does not occur with unsecured sockets).

Stopgap solution is simply to append \t characters to the message, which is stripped by Q Server. If after stripping \t characters, the message is null, then it does not reply and keeps reading.

Complete solution would involve not sending any messages until a defined END character is seen from instream (in Q Server).

**Quiz Page**

Fairly trivial, going next/prev goes to next/previous question, which is loaded from the list of Question objects. Submit prompts the QuestionClient object to query the Q server with a MARK\_Q request.

If this returns true, then set the marks for this question to be the number of requests. If false is returned, reduce attempt count by 1.

Go through BitSet implementation and how this is checked on every submit request → If all bits are 0, then set session to complete and only display final results page.

Exploits/Failures

A fairly big one. I neglected to check the user input of programming questions (which are written in python) to disallow the use of certain functions such as opening sockets, subprocess, opening files, etc.

Could blacklist all such exploitable python functions – i.e. eval, exec, open, tarfile, etc.

<https://stackoverflow.com/questions/4207485/exploitable-python-functions>

Consolation is that the code is executed on the Testing Server, thus, the Question Server is immune from such remote code injection style attacks.

It’s also possible to edit the HTML to recreate the submit button (after it is removed), or simply keep an older page in the tab and hitting submit. This will reset the Question Mark to 0, but the total mark is unaffected, and thus this is considered a minor graphical bug.

# Question Server

* Although not specified by project, also has a login system for further security and authentication (although this should already be covered by either having the Question Server and Testing Server on the same local network with the question port blocked from sending packets beyond the network boundary)
  + Unfortunately in this case usernames/passwords are held as plaintext – these are intended to be held as cryptographic hashes similar to the TestingServer.
* When initialised, loads username/passwords and questions data
* Uses threading to solve multiple clients connecting to it
  + Choice between blocking I/O – single client OR non-blocking I/O OR using polling – didn’t implement this as project didn’t specify need for many Testing Servers to connect to one Question Server.
  + For the scale of our project its assumed that this has a negligable impact on system resources, since we shouldn’t have many Q Server clients.
* List of string Lists representing questions are loaded from json files in “store” directory. These have a fairly intuitive and easy to understand format.
* T2Q protocol
  + INIT\_SESS – Tell QuestionServer to init a session, which also generates the questions. Need to provide the session ID, as well as a username/password for the Q Server.
    - In: IS | Session# | Q\_limit | username | password
    - Out: OK or ERR
  + GET\_Q – Get a question from the QuestionServer. Provide session ID and Question ID.
    - QG | Session# | Q#
    - Out: question|a|b|c|d OR question OR ERR
      * Note: response determines question type (mcq or programming).
  + MARK\_Q - Mark attempt. Provide Session ID, Question ID and answer in string form.
    - QM | Session# | Q# | Answer
    - Out: true OR false OR ERR
* Does not check/authenticate clients connecting to it, but only creates a Session if authenticated user/pwd is passed with INIT SESS.
  + Assume that Q Server and T Server are on the same local network…
  + Could require Client Authentication – i.e. Client would need certificate also signed by Root Authority.
  + I think this solution is relatively harmless? - exploits related to open sockets.

Weakness of protocol: The Need to send session information with each request.

Strength: Actual username information is not revealed to Question Server – only testing server has the connection between usernames and sessions.

Question Server and Testing Server are fully autonomous and only share what is necessary.

In theory, a further optimisaton would be if answers were also passed to Testing Server along with Questions, no need for Question Server to handle marking as well – and might be better in case the Question Server terminates unexpectedly - but this was in the project specifications, so this was left in.

# Problems

* Currently if either server catastrophically fails (either of the servers), the only way to reset it is to kill both servers and start again. This also results in a loss of all state data (quiz and marks).
  + There was planned to be a state saving process to save serialized final marks and quiz completion data to JSON, which could be loaded when the TestingServer first initialises, this was never completed however.
* As previously mentioned, Sessions can be easily infiltrated if somebody were to share their Session ID.
* Embarasssingly I forgot to dump the final marks for each user in a text file as intended, the method exists but I forgot to complete its implementation.
* Passwords for Question Server authentication are stored in plaintext (although User authentication for Testing Server are not). Was intended to be similarly hashed and salted but not finished.
* The behaviour when invalid queries are sent to the Question Server is not well-defined and processed by Testing Server. i.e. it returns an err message, but currently the Testing Server does not handle any such ERR case (Simply causes a generic error in browser).
* Fundamental problem with way instream/outstream data is processed, SHOULD read until null but assumes all data arrives 100% reliably, which is not always the case. Not fixed due to time constraint.
* User-run code in the dynamic programming questions is NOT checked, allowing user to run dangerous operations such as running subprocess, or being able to open network sockets. This could be trivially solved by blacklisting the keyword “import”, or a tighter implementation might involve a defined blacklist or whitelist.
* Solution is not very scallable – designed very specifically for this scenario. Uses threading to solve multiple client connections for Q Server. Would be better to use select() - Schronous I/O multiplexing, polling of descriptors instead.
* Using static HTML whilst simple also makes it very simple to modify HTML files to produce invalid responses or to hijack into another user’s session.
* Not clear how certificates/keys would be shared between TestingServer/QuestionServer. Right now assumes some kind of out-of-band communication of keys. This could be done in the program itself (although ironically perhaps be less secure?)
* Program has a lot of inefficient string processing – Kind of abused strings a lot. Could be tidied up – Notably the processing of different requests on the T→Q protocol uses a lot of string comparison.
* No way for students to view correct answer of question/feedback mechanism. Once quiz is completed, no quiz review mechanism.
* TestingServer eats memory – due to need to store Session data, not “freeing” question information once the quiz is completed.

# Extension Ideas

# Fixing above mentioned issues – i.e. using StringBuilder instead of adding immutable strings. - State loading mechanism from serialized data. - Using URL rewriting rather than hidden HTML field.

# Probably would if I had the choice written the whole project in one language

# Prettier UX for the Quiz interface – Using CSS and perhaps dynamic website vs static html

# GUI for TestingServer and QuestionServer – Able to change settings (number of questions (mcq/programming)) through GUI

# Changing IPBuckets – which is fairly useless - to instead be a leaking buckets implementation for password attempts (3 every 30 minutes)

# Implementing Timer for Quiz (i.e. if Quiz is not completed in X amount of time, timer expires and quiz is considered completed).

# For passwords – perhaps reading from haveIbeenpwned cracked password hash database <https://www.troyhunt.com/introducing-306-million-freely-downloadable-pwned-passwords/> to ensure users cannot use bad passwords. Would just need to pass HTTP GET requests to the API link and get response.

# Randomise quiz questions and answers more. Perhaps a question could have 10 possible answers (7 bad, 3 good) and randomise one good answer with 3 bad answers, or create a new class of arithmetic questions with fully randomised questions/answers.

# [Creating an Online Quiz Application Using JSP Servlet | Edureka Blog](https://www.edureka.co/blog/creating-an-online-quiz-application-using-jsp-servlet/) [YouTube](https://www.youtube.com/watch?v=mStnzIEprH8) [Secure Sockets - Java Sockets Tutorial 06 - YouTube](https://www.youtube.com/watch?v=l4_JIIrMhIQ)

# [SSL with Java example using simple client server echo app - YouTube](https://www.youtube.com/watch?v=VSi3KFlVAbE) [Secure programming with the OpenSSL API](https://www.ibm.com/developerworks/linux/library/l-openssl/index.html)

# [SSL Programming Tutorial](http://h41379.www4.hpe.com/doc/83final/ba554_90007/ch04s03.html) [Blocking vs. non-blocking sockets](https://www.scottklement.com/rpg/socktut/nonblocking.html)

# [Java NIO SocketChannel](http://tutorials.jenkov.com/java-nio/socketchannel.html)

# [Handling many sockets at once using select()](https://www.scottklement.com/rpg/socktut/selectserver.html)

# [Using SSL with Non-Blocking IO - O&apos;Reilly Media](http://www.onjava.com/pub/a/onjava/2004/11/03/ssl-nio.html?page=last&x-order=date)

# [Secure Your Sockets with JSSE - O&apos;Reilly Media](http://www.onjava.com/pub/a/onjava/2001/05/03/java_security.html)

# [ssl - Simple Java HTTPS server - Stack Overflow](https://stackoverflow.com/questions/2308479/simple-java-https-server)

# [com.sun.net.httpserver (Java HTTP Server )](https://docs.oracle.com/javase/8/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/package-summary.html)

# [Create a Simple Web Server in Java (1) - HTTP Server - CodeProject](https://www.codeproject.com/tips/1040097/create-simple-http-server-in-java) [Login form in Servlet - javatpoint](https://www.javatpoint.com/example-of-login-form-in-servlet)

# [Session Management in Java - HttpServlet, Cookies, URL Rewriting - JournalDev](https://www.journaldev.com/1907/java-session-management-servlet-httpsession-url-rewriting)

# [encryption - Choosing a session ID algorithm for a client-server relationship - Information Security Stack Exchange](https://security.stackexchange.com/questions/24850/choosing-a-session-id-algorithm-for-a-client-server-relationship)

# [Hidden form field for Session Management | Servlet Tutorial | Studytonight](https://www.studytonight.com/servlet/hidden-form-field.php) [simple HTTP server in Java using only Java SE API - Stack Overflow](https://stackoverflow.com/questions/3732109/simple-http-server-in-java-using-only-java-se-api)

# [Secure Sockets in Java | Bits and Bytes](https://pa55word.wordpress.com/2007/11/23/secure-sockets-in-java/) [JSSE Reference Guide for Java SE](https://docs.oracle.com/javase/7/docs/technotes/guides/security/jsse/JSSERefGuide.html)

# [TCP Performance problems caused by interaction between Nagle’s Algorithm and Delayed ACK](http://www.stuartcheshire.org/papers/NagleDelayedAck/)

# [Java Server SSL Socket and Python Client SSL socket - problems when server send messages - Stack Overflow](https://stackoverflow.com/questions/11787467/java-server-ssl-socket-and-python-client-ssl-socket-problems-when-server-send)