The following tests should be executed to verify functionality of the broken hash application. These tests are not to be taken as the whole list of tests that could be executed. These are the bare minimum tests to cover each of the application requirements.

For information on set up, requirements, and sample commands, see the appendices.

**Tests**

Generate a password hash

This is a simple straightforward test. It verifies that the application listens to HTML requests, it outputs a job number then waits 5 seconds, and then after the command runs you can request the generated hash by job number. It also verifies that you can configure the port.

1. Send the application a password to hash; use the port set up previously (usually 8088).
2. You should receive a job number immediately.
3. After waiting 5 seconds, use the job number you received to retrieve the hash from the server.
4. You should receive a hash in response.

Send simultaneous requests

This test verifies that the application can handle more than one request at a time.

1. Ready two simultaneous hash requests in separate terminal windows.
2. Send both requests as simultaneously as possible.
3. Retrieve the hash information based off job numbers.
4. Both hash requests should have been processed normally and returned on request.

Get a non-existent job number

In this test we check that a request with a non-existent job number fails gracefully.

1. Get a hash for a job number that has not been used yet.
2. Examine the resulting error message. It should tell you “Hash not found”

SHA512

According to requirements, the SHA512 hash algorithm should be used. We test that in this test case.

1. Request that the application create a hashed version of a password you provide.
2. Retrieve the hash.
3. Use the same original text to generate a hash using an online sha512 generator
4. Compare the two hashes.

Stats

We verify that the necessary stats (number of jobs and time worked) are available by API. Also, we verify that extraneous information is ignored.

1. Send a stats request to the application.
2. Compare the resulting information with the number of requests sent.
3. Send another stats request, but include data in the request. For instance, you could reuse the hash request but change “hash” in the url to “stats”.
4. Stats will just return the normal stats; the input will have been rejected.

Shutdown

This test verifies that the application shuts down gracefully.

1. Ready a shutdown request and two password hash requests in their own terminal windows. Do not send yet.
2. Send the first password hash request. Immediately initiate the shutdown.
3. Send the second hash request while shutdown is in progress. Verify that the request is refused.
4. Verify that the password hash request completes before shutdown completes.

**Appendix I - Setup**

Obtaining the Password Hashing Application, execution and Operating System Specific Instructions

We are storing the password hashing application in a public S3 bucket, you can get it in the following manner:

*$ wget --no-check-certificate --no-proxy ‘*[*https://s3.amazonaws.com/qa-broken-hashserve/broken-hashserve.tgz*](https://s3.amazonaws.com/qa-broken-hashserve/broken-hashserve.tgz)*’*

The broken-hashserve.tgz archive contains binaries for Linux, Windows & MacOSX operating systems.  Unpack and use the binary corresponding to your OS of choice.  We are certain that the application will not run amok and trash your workstation/laptop; that being said, you should always run untrusted binaries in a virtual environment for a number of reasons :)

You ***must*** set a PORT environment variable before executing the application, it will crash otherwise.

*$ export PORT=8088*

On modern Windows systems with PowerShell 3.x, curl/wget/iwr are just aliases for Invoke-WebRequest it has equivalent curl/wget/iwr parameters and output, they are just named different things.

We tested the password hashing application on the following :

* Ubuntu 16.04
* Mac OS X - Sierra, High Sierra
* Windows 10

**Appendix II – Requirements**

The following is the requirements specification that was used in building the password hashing application.  It describes what the application ***should*** do.

* When launched, the application should wait for http connections
* It should answer on the PORT specified in the PORT environment variable.
* It should support three endpoints:
  + A POST to /hash should accept a password.  It should return a job identifier immediately.  It should then wait 5 seconds and compute the password hash.  The hashing algorithm should be SHA512.
  + A GET to /hash should accept a job identifier.  It should return the base64 encoded password hash for the corresponding POST request.
  + A GET to /stats should accept no data.  It should return a JSON data structure for the total hash requests since the server started and the average tie of a hash request in milliseconds.
* The software should be able to process multiple connections simultaneously.
* The software should support a graceful shutdown request.  Meaning, it should allow any in-flight password hashing to complete, reject any new requests, respond with a *200* and shutdown.
* No additional password requests should be allowed when shutdown is pending

**Appendix III – Sample input**

You can interact/test the application using curl.  The following are examples that would/should generate similar returns - the job identifier does not need to conform to a specification.

* Post to the /hash endpoint

*$ curl -X POST -H "application/json" -d '{"password":"angrymonkey"}'* [*http://127.0.0.1:8088/hash*](http://127.0.0.1:8088/hash)

*> 42*

* Get the base64 encoded password

*$ curl -H "application/json"* [*http://127.0.0.1:8088/hash/1*](http://127.0.0.1:8088/hash/1)

*> zHkbvZDdwYYiDnwtDdv/FIWvcy1sKCb7qi7Nu8Q8Cd/MqjQeyCI0pWKDGp74A1g==*

* Get the stats

*$ curl* [*http://127.0.0.1:8088/stats*](http://127.0.0.1:8088/stats)

*> {"TotalRequests":3,"AverageTime":5004625}*

* Shutdown

*$ curl -X POST -d ‘shutdown’* [*http://127.0.0.1:8088/hash*](http://127.0.0.1:8088/hash)

*> 200 Empty Response*