SecureSpace: Astronaut Health Monitoring Application

Post-graduate Computer Science programs Group Project for Secure Software Development - University of Essex Online

Team Members: *Bradley Graham, Michael Sammueller, Tomas Mestanza & Rachel Doherty*

Table of Contents

How to run the program

In order to install all the dependencies required for this project run the command:

```
pip install -r requirements.txt
```

This will install all of the packages in the requirements.txt file.

Three demo accounts have been provided, each with different privileges. These can be altered in the application/classes/authseeder.py file

1. Username: demosuperadmin, Password: password123, Secret Phrase: demo, uuid: 100
2. Username: demomoderator, Password: password123, Secret Phrase: demo, uuid: 101
3. Username: demoastronaut, Password: password123, Secret Phrase: demo, uuid: 102

Tests can be found in the application/tests directory.

Description

This application is intended for use by astronauts to record health indicators related to their physical fitness and mental health during a space mission. It also models the measurement of ambient temperature and radiation exposure via onboard sensors.

The project domain is the International Space Station (ISS), which is a collaborative international space travel and research program between 15 governments, overseen and facilitated by NASA. The application is

intended to support the health and safety of astronauts aboard the ISS during a mission in order to prevent crew health issues jeopardising the success of a mission. The application responds to a report by the ISS Safety Task Force (ISSTF) which deemed crew health issues to present a major risk to the successful operation of the ISS (ISSTF, 2007).

Design Principles

The application is accessed via a Command Line Interface (CLI) and allows the User to perform various CRUD actions, based on predefined role-based permission. The project brief was to implement secure software development principles into the application design.

The project has been deliberately developed in a modular way using object-oriented coding principles and with a 'microservice' architecture in mind, which aims to produce modules of code which encapsulate their own methods and functionality and interact only with necessary modules within the program in order to improve modifiability, testability, scalability and code efficiency. No networks or virtualisation have been implemented in this version.

The project implements secure designs techniques including encryption, input sanitisation, role-based access control, database normalisation, action and event logging, removal of echo on input, uuids, and password complexity enforcement.

The database application is designed to be accessible via a single terminal onboard the ISS and does not require onboard internet network connectivity. User data is accessible only for the mission in question for the crew members onboard that particular mission. User data and profiles will be deleted following each mission and set up again for the crew of the following mission. This enclosed deployment and limited usership maintains the application's low attack surface.

Passwords and some user data are encrypted before being stored in the database, however some database records do remain unencrypted. The unencrypted records do not reveal personal information which could elevate permissions. The superadmin has most permissions. If superadmin access details were leaked, there would be no way to access the database without access to the onboard terminal which is located in space, onboard the ISS.

According to the principles of Agile and Lean software development, functionality should only be implemented if it brings value to the product and enhances the architectural quality of the system (Alahyari et al., 2017). For this reason and with the domain in mind, SecureSpace consider it counterintuitive to encrypt the entire database at this point, which slows down application performance.

On first-time deployment of the application and database intialisation, a user with superadmin rights will be able to add user profiles. It is assumed that usernames and passwords will be assigned by a superadmin to the individual mission crew members and shared in an 'analog' environment. No functionality to distribute login data via email or other means has been incorporated, which consequentially can be considered an additional layer of security.

Application features & implemented technologies

- **Languages:** Python / SQL
- **Database:** SQLite3
- **Interface:** Command Line

- **Network requirements**: Access via a local terminal (no web access required)

The program is written in Python3 and uses the following built-in libraries: **re, logging, unittest, getpass, threading, pip, json, datetime, random, sys, uuid, cryptography, unittest, abc, os**

External tools & libraries

- **bcrypt** - for password hashing
- **Pylama** - for checking code quality
- **Bandit** - to check for common security flaws
- **Pytest Security** - to write security tests in python
- **coverage** - to check test coverage of code

Database

This solution makes use of a relational database to store data relevant to the system.

Normalisation has been applied to the database design. Applying the normalisation theory aims to reduce data redundancy and avoid potential problems when performing operations on the database (Eessaar, 2016).

As a consequence of applying normalisation, consistency is improved, and maintenance is reduced in complexity.

UUIDs are use to avoid reveal actual user IDs. We kept the numeric IDs to be able to sort as UUIDs do not allow sorting.

Database Considerations

In general, constraints are not implemented at this stage but applying the correct SQL statements will generate the necessary structure to handle deleting associated records.

Minimum indexation has been applied to ensure the queries run quickly.

Security Features Checklist

This project is based on secure design principles, following key vulnerabilites identified in the "OWASP Top 10 Application Security Risks" (2017). The following checklist outlines the implemented security features:

OWASP Top 10: Planned vs. Implemented measures

A1: 2017 Injection

- [x] Sanitize input (filter keywords and special characters) for all roles

A2:2017 Broken Authentication

- [x] multi-factor authentication (secret phrase)
- [ ] time-limited set-up passwords - *not necessary, password distribution not in scope*
- [x] password length and complexity requirements (min. 8/ max. 64 characters; allow most characters)
- [x] limit, log and alert all failed login attempts

A3:2017 Sensitive Data Exposure

- ☐ encrypt database records - *unnecessary for system functionality/ security - see above*
- ☑ store passwords using strong salted hashing functions (bcrypt)
- ☐ penetration testing (incl. link to evidence when done)
- ☑ delete sensitive info after mission ends - included, but cannot be simulated by system, see data deletion policy

A5:2017 Broken Access Control

- ☑ robust roles
- ☑ no role-inheritance

A6:2017 Security Misconfiguration

- ☑ only implement necessary libraries
- ☑ only implement OWASP-approved or known libraries
- ☑ ensure correct configuration of all technologies through testing

A9:2017 Using Components with Known Vulnerabilities

- ☑ implement a tool that checks for dependencies and security vulnerabilities (bandit) (see test result)

(A10:2017 Insufficient Logging & Monitoring)

- ☑ implement an effective monitoring and alert system for auditable actions
- ☐ store log records in a dedicated database with restrictive commands - *not implemented, time limitations*
- ☑ encrypt logs

List of changes - Design Document vs. Implementation

1. Class 'CommonActions'

Original System Design: The 'CommonActions' class was originally planned as a class to handle all functions and methods relating to user actions.

Final Implementation: The class has been renamed to 'ActionController' and will act as an interface between the command line and the system objects. It will therefore always call another object's version of a method. This way, the application is more scalable as services can be replaced.

2. File Downloads

Original System Design: Originally, we would allow users to download their health records as data files.

Final Implementation: File download has been implemented with the resulting file in .txt format.

3. Role / Permissions mapping

Original System Design:

1. Superadmin permissions: create user, assign roles and privileges, execute SQL queries for database management
2. Moderator permissions: approve user, delete user profile and data
3. Astronaut permissions: update and view health records, download data files

Final Implementation: The planned roles did not cover system functionality. We also needed to map permissions to user actions / roles explicity in order to ensure that role-based access controls (RBAC) was implemented robustly. The superadmin should perform all actions.

The updated role matrix maped to permissions and roles is shown in the table below:

UserAction / Permission Name	***Superadmin***	***Moderator***	***Astronaut***
Add New User / create-user	[x]	[ ]	[ ]
Delete User / delete-user	[x]	[ ]	[ ]
View All Users / view-all-users	[x]	[ ]	[ ]
View User Details / view-user-details	[x]	[ ]	[ ]
Edit User Details / update-user-details	[x]	[ ]	[ ]
Add Health Record / add-health-record	[x]	[x]	[ ]
Add Own Health Record / add-health-record	[x]	[x]	[x]
View User Health Records / view-user-health-records	[x]	[x]	[ ]
View Own Health Records / view-own-health-records	[x]	[x]	[x]
Edit User Health Record / update-user-health-record	[x]	[ ]	[ ]
Delete User Health Records / delete-user-health-records	[x]	[ ]	[ ]
View Temperature / view-temperature	[x]	[x]	[x]
View Radiation level / view-radiation	[x]	[x]	[x]
Change password / change-password	[x]	[x]	[x]
Download Health Record / download-record	[x]	[x]	[x]

Testing

Integration and unit testing has been implemented throughout the system build process to periodically verify that system features are working as intended (see here). Whitebox testing has done throughout too.

GDPR

The application is developed according to UK GDPR.

The lawful basis for data processing is subject to:

1. the consent of the data subjects;

2. the vital interests of crew health as a key ISS vulnerability (IISTF, 2007);
3. the continuation of ISS operations, as outlined in the legal framework of the ISS (IISTF, 2007)

It is assumed that astronaut consent to data processing is sought before any mission commences by that respective national space travel authority (e.g. the UK Space Agency, the Japan Aerospace Exploration Agency etc.).

Data Security Statement

We have prioritised the security and privacy of users' personal data. This application is developed with stringent measures in place to protect personal information against the vulnerabilities outlined by the OWASP Top 10 common security risks framework (see "Security Features Checklist" for detailed protective measures).

Data Privacy Policy

Personal data is collected for the intended purpose only (astronaut health monitoring for the duration of a single ISS mission) and will not be shared with 3rd parties outside of NASA/ ISS operations. No third parties will have database access without having access to the onboard terminal, which is in space on the ISS. Remote access to the program is not possible as it is neither web-based nor exposes external APIs. Third party access to personal data is therefore only for athorised mission/ crew members who have superadmin system access.

Data Deletion Policy

The intended system implementation is that user profiles and user data will be available for the duration of a single ISS mission only. All personal data and user profiles are to be deleted from the database immediately following a mission.

Software Development Lifecycle

The team has made us of communication tools such as slack, trello; as well as github as the main repositiory. Changes were made incrementally and peer reviewed via pull requests.

References

Alahyari, H., Svensson, R. B. & Gorschek, T. (2017) A study of value in agile software development organizations. Journal of Systems and Software 125, 271-288, DOI: https://doi.org/10.1016/j.jss.2016.12.007

Eessaar, E. (2016) The Database Normalization Theory and the Theory of Normalized Systems: Finding a Common Ground. Baltic J. Modern Computing 4(1): 5-33. Available from: https://www.researchgate.net/publication/297731569_The_Database_Normalization_Theory_and_the_Theory_of_Normalized_Systems_Finding_a_Common_Ground [Accessed 21 June 2023].

IISTF (2007) Final Report of the International Space Station Independent Safety Task Force. NASA, pp 1-111.