# 1 Overview of the Application:

This application allows the user to submit security vulnerability reports for further analysis. Since most of the breaches occur due to human error or simple misconfiguration, the application makes sure that the user's input is sanitised, before it gets processed by the application. The application also makes sure that any sensitive data submitted by the user will be encrypted to ensure confidentiality. Further enhancements to ensure integrity could potentially include hashing and admin trail, and to address any availability concerns, the application can be hosted on multiple physical, or virtual hosts, automatically failing over in case of a host failure.

## 1.1 Instructions on Execution:

The application requires several libraries to be included and installed. These can be either installed individually or by utilising the provided requirements.txt file and then running

```pip install -r requirements.txt``` in the terminal.

The code was written in Python version 3.9, but depending on the interpreter, the version used by the system might need to be downgraded to 3.8. The latest version of pip needs to be installed to support certain libraries – this version is 23.1

1. Run the App.py file; this can be by running the command ```python App.py``` in the terminal.
2. The application will run on the local host (http://127.0.0.1:5000); navigate to this address in the browser to access the app.
3. To register the new user, go to step 4 - "Registration", to log in as an existing user, go to step 5 - "Login"
4. Registration: Provide an e-mail, and a strong password and accept the policies
5. Provide an existing email address and password
6. For any account/profile-specific configuration, go to "Account Details", where you can:
    a) Permanently Delete the Account
    b) Change your details, including an e-mail address, name, and password
    c) View secure messages sent by the viewer of the report
    d) View existing, submitted security reports
7. To submit a report, click on "Submit Report"
8. Submit Report: Provide a type of vulnerability, explanation, and reason as well as any associated domains or IP addresses
9. To edit an existing report, either access it from the main dashboard/homepage or account details (as described above in 6d)

## 1.2 Default Logins:

The default users are as follows:

| Email | Password | Role |
|---|---|---|
| admin@ssdproj.com | admin@1234 | Admin |
| user@ssdproj.com | user@1234 | User |
| user2@ssdproj.com | user2@1234 | Admin |

### 1.3 Resetting the Database to Default State:

A helper utility script ```initialize_db.py``` is provided which just deletes all data records from the database (users, reports and messages) and then adds in a set of default users (as per the list above) and some placeholder demo reports and messages. It can be used by running ```python initialize_db.py``` at the command line while in the application folder.

## 2 Additional Information, limitations and potential improvements:

- The administrator (user.id==1) is an internally created account populated into the database from the backend; this admin has the ability to elevate user accounts to admin level when needed; the account can't be deleted unlike other accounts; it's role can't be changed; and it's details can only be edited by this user themselves
- One significant improvement can be the inclusion of a more comprehensive access control system that defines exactly what tasks each user can and can't perform. Currently:
  - The "User" can create reports, edit and view their reports, post messages pertaining to their reports, delete their messages, edit their own account details, and delete their account.
  - The "Admin" (any user that is assigned the "Admin" role with an account that isn't admin@email.com) can do everything that users can do, as well as viewing/editing/deleting all reports, messages and accounts.

- A more comprehensive access control system coupled with enabling two-factor authentication (2FA), especially for sensitive tasks such as editing and deleting of reports, messages or accounts will help secure the system further; e.g. when an admin user tries to edit/delete a report, the super admin should receive a notification and accept the action. 2FA was not included as this would make testing and demonstrating the app significantly more challenging in this demo; however, sample code for this feature is provided at the bottom of App.py class
- Any permanently deleted user will have their personal details deleted but their email, reports and messages will remain in the database for a specific amount of time depending on the regulations (anywhere between 1 day to several years)
- Any e-mail-specific operations are not configured in this version of the application, due to the additional complexity required for processing mailboxes and additional encryption necessary

- The application is hosted on the local machine, in a production environment, the application would be hosted and publicly accessible – this is not configured in the current version of the application due to additional complexity with domain registration, DNS, public certificate hosting, as well as additional costs.
- Any sensitive information is encrypted using the SHA algorithm and handled using the Fernet library
- In a production environment, a secure dedicated database management system such as PostgreSQL or MySQL, separate from the web application, should be used. For this demo, we did not have a server to host a dedicated database and we therefore used an SQLite database. The database interface used is the SQLAlchemy library which means that it should be very easy and seamless to change to another database management system such as MySQL or PostgreSQL. The change of database can be made in the App.py file on line 24
- The application features the use of cookies, allowing the user’s session to be remembered
- The application features the use of a limiter, limiting the number of requests a user can make per minute/second, to protect against potential memory overflow attacks (DoS/DDoS)

## 3 Differences of Final Solution to Initial Design Document: (424 words)

Below are summarised the differences of the final implementation with the design in the initial design document, where the differences are compared to the use case diagrams, the class diagram and the sequence diagram in the initial design.

### 3.1 Use Case Diagrams:

With regard to the use case diagrams, the initial design document and the final implemented system match almost perfectly, where all the tasks shown in the initial design document are provided to the relevant users. The exceptions are:

- Editing messages; this was omitted given the ease of posting new messages and given that CRUD capabilities were demonstrated with Users and Reports.
- User deactivation; user deactivation in the final system is packaged as deletion but is actually just a deactivation of the user; it was decided to avoid complete deletion of the user at this stage.
- User activation; user reactivation only takes place by means of a user re-registering on the site (after their account has been deleted).

Also, the use case diagrams in the initial design document only specify two types of users; an Admin and User. In the final system, there is a Super Admin user (user with user.id==1 and role=="Admin"), Admin users (users with user.id!=1 and role=="Admin") and Users (users with role=="User"). While Admin users can do almost everything that the super admin can do, they can't edit the super admin account's details or password; furthermore, the super admin account can't be deleted (even by the super admin themselves), neither can it's role be changed.

## 3.2 Class Diagram:

The design of the final system differs very significantly with the design shown in the class diagram in the design document. Ultimately, we ended up with three classes: one for the User, one for Reports and one for Messages, which reflects the design of the database as well. The functions used are mostly those of Python Flask itself.

## 3.3 Sequence Diagram:

The sequence diagram also reflects very closely the implementation of the final system in that almost all tasks in the diagram are represented in the final system and function according to the way that the diagram initially proposed. The exceptions are below:

- 2FA was not included in the final implementation
- Password recovery was not included
- While all of the data generated by the user is summarily displayed on the user's account page, there is currently no option to email this data to the user or to download it, as these actions both present a very significant security risk (given that the information will exclusively contain information about vulnerabilities in Government software).

# 4 Bibliography:

docs.python.org. (n.d.). json — JSON encoder and decoder — Python 3.8.3rc1 documentation. [online] Available at: https://docs.python.org/3/library/json.html.

flask.palletsprojects.com. (n.d.). Welcome to Flask — Flask Documentation (2.2.x). [online] Available at: https://flask.palletsprojects.com/en/2.2.x/.

flask-sqlalchemy.palletsprojects.com. (n.d.). Flask-SQLAlchemy — Flask-SQLAlchemy Documentation (3.0.x). [online] Available at: https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/

Frazier, M. (n.d.). Flask-Login: User authentication and session management for Flask. [online] PyPI. Available at: https://pypi.org/project/Flask-Login/

Harihar, A. (n.d.). wtforms-validators: Validators for wtforms package. [online] PyPI. Available at: https://pypi.org/project/wtforms-validators/

Jacob, D. (n.d.). Flask-WTF: Form rendering, validation, and CSRF protection for Flask with WTForms. [online] PyPI. Available at: https://pypi.org/project/Flask-WTF/.

O’Connor, P. (n.d.). RUST: Ribo-Seq Unit Step Transformation. [online] PyPI. Available at: https://pypi.org/project/RUST/

O’Mahony, J. (n.d.). evervault: Evervault SDK - Encrypt. [online] PyPI. Available at: https://pypi.org/project/evervault/0.1.0/

PyPI. (2019). cryptography. [online] Available at: https://pypi.org/project/cryptography/.

PyPI. (n.d.). WTForms: Form validation and rendering for Python web development. [online] Available at: https://pypi.org/project/WTForms/

Python Software Foundation (2002). Datetime — Basic Date and Time Types — Python 3.7.2 Documentation. [online] Python.org. Available at: https://docs.python.org/3/library/datetime.html.

Saifee, A.-A. (n.d.). Flask-Limiter: Rate limiting for flask applications. [online] PyPI. Available at: https://pypi.org/project/Flask-Limiter/

Ueda, T. (n.d.). python-form: A package for communicating with FORM. [online] PyPI. Available at: https://pypi.org/project/python-form/

Vartanyan, M. (n.d.). password-strength 0.0.3.post2 : Password strength and validation. [online] PyPI. Available at: https://pypi.org/project/password-strength/.

werkzeug.palletsprojects.com. (n.d.). Utilities — Werkzeug Documentation (2.2.x). [online] Available at: https://werkzeug.palletsprojects.com/en/2.2.x/utils/