-Development Process

We completed the project in two phases a design and implementaion phase.

During the design phase we consdidered the functional and non-functional requirements of the the system and domain. We then considered the security implications of these features.

-Security Principles

We wanted to follow the best practice advise of academics and leading experts. We choose the open web application security project aswell as Saltzer and Schroeder Security Principles as important references for guidence.

-Attack Surfaces

During the design phase - and development of the design document- we considered the attack surfaces of our design. We identified the database, REST API, command-line interface, authticaions systems as well as Python itself and the libarys used as attack surfaces worth attention.

At this point in the project we docuemnted mitigations we thought apporpriate . When we reached the implementation stage we re-evaluated this result and used it as the starting point for implemting security features.

-System Architecture

Our system archetecture uses a client server design with a 3 tier layered structure. Our client applications form the interface layer. The server implements the business and datastore layers.

-Relation Class Diagram

In our design we used UML modeling and we then implemnted this design. Here we can see how our class diagrams mapped to our code implementations. Its not one to one.

-Relation to Activity Diagram & ERD

Here we can see how our activity and entity relation ship diagrams aligned.

-Functional requirments breakdown

We are now going to breakdown of a few of the functional requirements of the system and explore the security mitigations associated.

For the functional requirement that 'Both users will have a different interface to access the features specific to them.' we realized we would need to implement a login mechanism with a privilige system.

For example, for the privlages, guided by Saltzer and Schroeder's Fail Safe Defaults principle we by default to denying ground controll staff extra priviages and only expose them if the admin account is used. We also hide users passwords to help prevent them leaking and we dont store users passwords longer than their current session.

-Functional requirements breakdown 2

For loging in we have implemented a simple multi-factor authentication we we verify the users emails when the register their account.

-Functional requirements breakdown 3

We implemented atomic operations in the data accesslayer. ATOMIC operations help to maintain the integrity of the data. In order for an operation to be ATOMIC it is important that if any part of the operation fails the whole operations does and that no fragment of data is inserted. We have ensured this by using the SQLites connection commit feature. It is only when the commit function is called that we write all the changes to memory. If any errors occur in our operation the commit will not be called and no changes will be made.

-Requirements Breakdown

Over the next few slides we will consider how the archetecture informed our security implementaion.

-Server logging and vulenratbility monitoring

Logging is important to the creation of a secure software systems. Logs can provide insight into the operation of the system and can aid debugging exersizes. We have dicided to logg all calls to the server to provide information about the systems usage.

We have implemented a vulenrablity assesment as part of the live system. The system routinly executes a vulnerability asssesment of the python packages used. This check involves comparisions with a vunlerability database. The result of this assesment is stored in a file called 'safteylog.txt'.

-Requirements Breakdown

We have made use the HTTPS secure communications protocol to securely comminicate over the network. This protocol uses a form of pulib private key encryption. Our client applications have access to a public certificate that signs our requests.

Access to many of the endpoints is further restricted to only users that have been granted an 'advanced privilege key'. This key is only transfered to the user when they provide a correct pair of email and password. The key is sent over the network secured by the use of HTTPS and stored for the duration of the session on the client.

The only two endpoints that do not require this key's authorization are the two endpoints required for the user to login.

-Data access layer

We have used SQL constraints in the data access layer to futher secure our datas integrity. The use of constraints seen here such as NOT NULL and forign id references ensures only valid data in inserted.

-Client

It is important to validate that the users input is within the 'accetable' range. In this code example we see that the users email input is validadted with a control structure that requires a server check of whether the account exists.

- DDOS Monitoring

Restfull interaces are vulnerable to denial of service attacks over the internet. We have a monitoring feature that displays in realtime the number of calls being made to the server. If this frequency of calls is extraordinarly high then we display a warning here to indicate an attack may be underway. We have set the threshold to 60 calls per 15 mins.

-SQL Injection attack prevention

We have used SQL paramatized querys so that all the users input is automaticaly esacped. This prevents malicous users sql querys from being excecuted on the database.

-Crytography

We have used cryptoraphy to help protect the database attack surface we identified. If the database is exposed the password contents is encrypted so that is is not readily accesable. The contents can still be revealed if the attackers use a rainbow table to decrypt the stolen password information. The encryption however adds an extra layer of security and will at the very minimum provide additional time for other mitigations to be taken - for example notifying users of the breach.