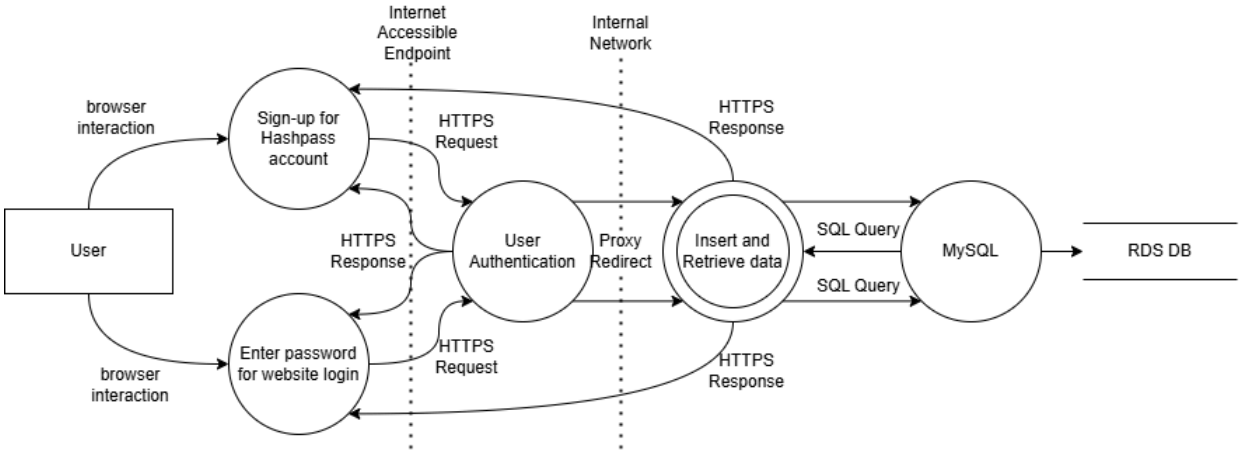
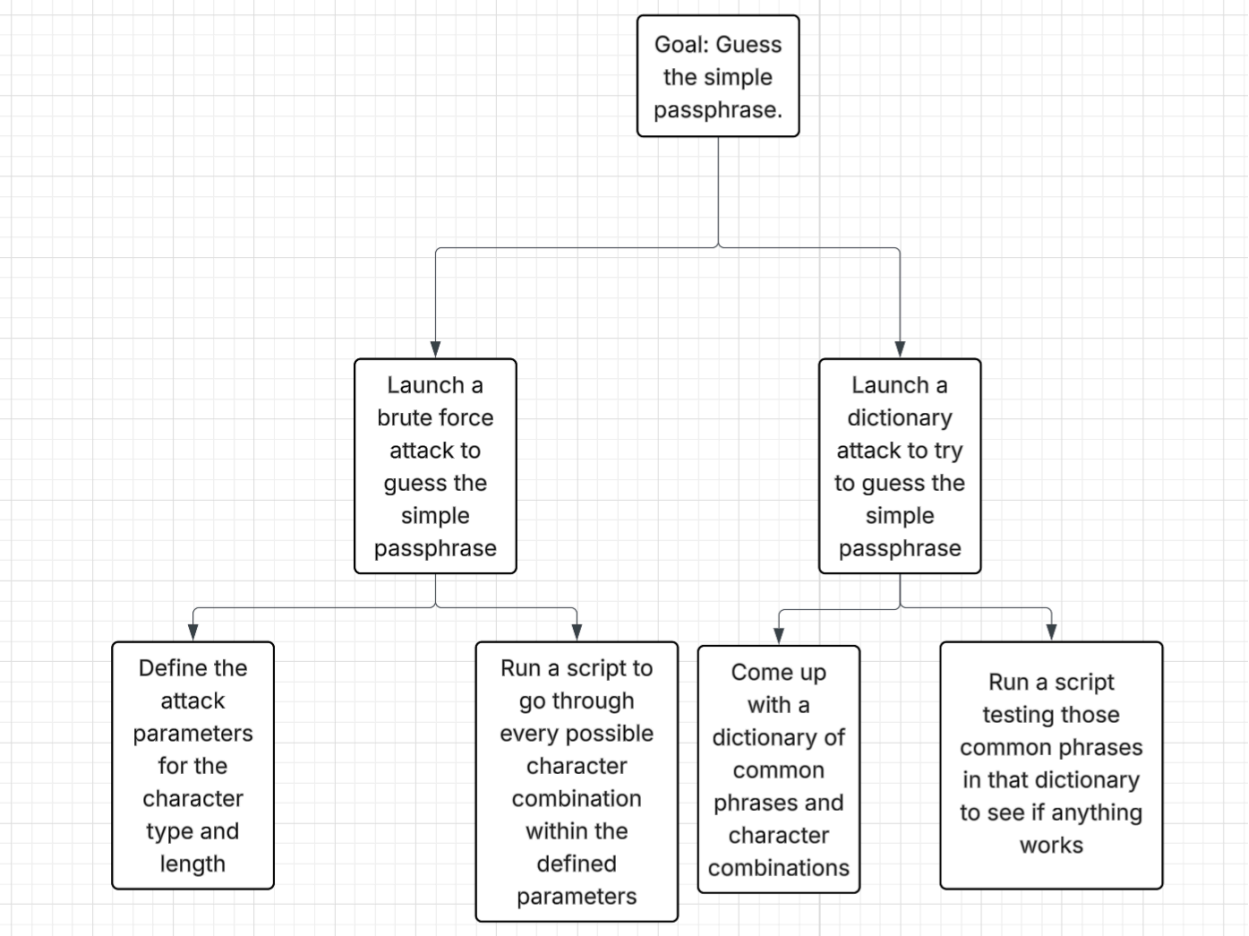
Security Design Review - Team 4

1. The figure below shows the data flow diagram with trust boundaries. The user uses the browser to sign-up and login on HashPass as well as enter their password to generate their password for a given site. The information provided by the user is hashed and sent to our backend using HTTPS responses crossing a test boundary from the user's device to our cloud backend which is open to the internet. From there, we have a user authentication point which verifies that the user is valid. This barrier does not ensure that the information being passed by a valid user is safe, thus there is another trust boundary between the authentication and database logic. The insertion and retrieval logic is what determines if the body of each request is valid. As this part of the logic is directly connected to the MySQL engine and our database, there is no trust boundary. The response from our backend whether from the user authentication or database handling logic will be in the form of a HTTPS response to the logic we have in the browser. The information that attackers would try to access would be the information in the database and potentially intercept the password entered by users to gain access to their Hashpass account and their website passwords. To avoid these areas of interest to attackers, we use HTTPS to ensure all internet traffic is encrypted, and that the information being sent is also hashed in case the encryption is broken. The database is set up such that it is accessed only from our database processes which requires the user to be authenticated. There is further validation in the database processes to ensure that an authenticated user cannot abuse their privileges to gain access to the information being stored.



2. One threat to our product is the threat of an attacker attempting to guess their way into an authentication either through a brute force attack or a dictionary attack. If somehow the attacker gains access to the user's browser on the user's computer, they can run a script to try to guess the simple passphrase to unlock the password vault. Another threat to our product is that the attacker may gain access to our database. The motivation for the first threat would be the most basic way to attack our password vault. It is the simplest form of attack. The motivation for gaining access to the database is that many password vaults store passwords in the database and gaining access to the database

may result in the attacker gaining access to the passwords. However, as explained in the next section, both these threats are mitigated in our product solution.



3. In order to mitigate the first threat laid out above is simply to follow industry standards for passphrase creation. While this is a pretty generic response, it is also specific to our project as we require 12 characters, including at least 1 number and 1 symbol. This can give us the comfort of safety as, according to the 2024 Hive Systems Passwords Table, it will take a brute force attacker 164 million years to crack. These 12 characters can also protect against a dictionary attack as the safe recommended password length from Microsoft is at least 12 characters. In terms of our second threat laid out, it would be very difficult for an attacker to gain access to our database. If the attacker were to gain the credentials to our database, they would not be able to view it due to the Amazon VPC that was set up. The VPC restricts which instances are allowed to view the database. Even if an attacker were to gain access to the database no actual information would be leaked as the data in the database is encrypted with AES. The attacker would not know how to map the encrypted data back into plaintext without the user's passphrases, which is never stored in the database.