1. The Code Review Process

A Secure Code Review is a particular errand with the objective of distinguishing sorts of shortcomings that exist inside a given code base. The undertaking includes both manual and robotized survey of the hidden source code and recognizes explicit issues that might be agent of more extensive classes of shortcoming characteristic in the code. A Secure Code Survey doesn't endeavor to distinguish each issue in the code, however rather endeavors to recognize kinds of hazard inside the code with the end goal that moderation techniques can be contrived.

During the real audit, individuals from a survey group survey the application code for security issues and order the discoveries dependent on the shortcoming classifications (e.g., verification, approval, and so forth.). Each finding is allocated a hazard rating of High,Medium, Low, or Informational. These discoveries and the more extensive shortcoming classes that they speak to are introduced right now that the improvement group can use as the establishment for improving the general nature of the code base. It ought to be noticed that while the audit procedure will be as careful as conceivable in finding what's more, announcing security shortcomings, it isn't ensured to consistently locate each conceivable shortcoming. On the off chance that no issues are discovered, the survey doesn't certainly affirm that the application is 100-percent "hack confirmation."

A Secure Code Review is anything but a silver slug, however rather is a solid piece of a general hazard moderation program to secure an application.

2. Survey Summary

The protected code survey of the HPS wellness App application was finished on March 5, 2020 by an audit group comprising of our team. The audit was performed on code acquired from AWS production server credentials by means of email connection on March 5, 2020, and packaged under the record named hpsserver.jar.

A gathering between the audit group, was hung on March 9, 2020, at which time data about the code structure was exhibited alongside significant level outlines of how things like verification, information approval, and logging were executed in the code. This data was utilized by the audit group to define an arrangement for the approaching survey.The real audit included a manual examination of the Java code. Explicit source records were not doled out to singular individuals; rather, every individual from the survey group endeavored to survey the whole application. Every analyst recorded their particular discoveries inside a spreadsheet and doled out hazard levels as they felt fitting. Toward the finish of the audit, the group looked over the individual spreadsheets to analyze normal discoveries what's more, to perform bunch surveys of the remarkable discoveries. The particular discoveries are presented in the next section.

3. Discovering Summary

This area gives a rundown of the discoveries coming about because of this survey.

For this application, three elevated level issues were discovered identified with the regions of confirmation and information approval. One of the significant level issues coming about because of unvalidated assailant input being sent to the JSON parse() capacity could bring about discretionary directions being executed. Relieving activities ought to be considered. Some other medium and low issues have likewise been found. Subtleties are given underneath. The figures beneath graphically layout the audit group's discoveries by both classification and chance level.

1. There is a nohup document being created as a piece of the server logging process which is effectively acessible when server security is undermined.
2. The whole query is printed as log when running the server with the timestamp which would lead to data base schema being exposed.
3. All the backend data can be acessed using the app url even without login. This is limited to just the descriptor id and url.
4. All the login credentials for AWS database as well as the schemas can be easily acessed once user hacks/logs on to the server. Same is the case with the path to the directories where data is stored.
5. Unparseable date error when executing query for tracking habit id by date and sort descending.Please find attached query for reference ( select \* from wellness.habit\_tracking where habit\_id = ? ORDER BY scheduled\_date DESC) Error since 2019-10-22 12:17:15.151 server time.
6. Query (select \* from wellness.steps\_tracker where USER\_ID = ? and CAST (SCHEDULED\_DATE as DATE ) >= ? and CAST (SCHEDULED\_DATE as DATE ) <= ? order by SCHEDULED\_DATE) return the object with total number of steps as just yesterdays step even if the date selected is more than one day. Eg. Response that was given on 29-02-2020 at 8.38.43.636 server time for above query is as follows: {userRegisteredDate=2020-02-18 10:14:33.65052, caloriesConsumedData={totalCalories=1003.0, individualCaloriesReports=[{foodName=Wheat roti, totalCalories=595.0}, {foodName=Aloo ka bharta, totalCalories=306.0}, {foodName=Milk tea, totalCalories=102.0}]}, caloriesBurnedData={totalCaloriesBurned=851, caloriesBurnedIndividualData=[com.wellness.caloriestracker.vo.ActivitiesCalorieTrackingVO@4eec9250, com.wellness.caloriestracker.vo.ActivitiesCalorieTrackingVO@82b27a3, com.wellness.caloriestracker.vo.ActivitiesCalorieTrackingVO@7290bba0, com.wellness.caloriestracker.vo.ActivitiesCalorieTrackingVO@2b70c4d5]}, stepsTakenData={noOfSteps=1755, stepsHistory={25=4476, 26=6204, 27=7459, 28=5784, 29=1755}}}
7. select dietpatter0\_.id as id1\_0\_0\_, foodinterv1\_.id as id1\_3\_1\_, fooditem2\_.id as id1\_4\_2\_, locale\_foo3\_.id as id1\_7\_3\_, foodcatego4\_.id as id1\_2\_4\_, localefood5\_.id as id1\_6\_5\_, dietpatter0\_.created\_on\_date as created\_2\_0\_0\_, dietpatter0\_.created\_on\_time as created\_3\_0\_0\_, dietpatter0\_.food\_interval\_id as food\_int7\_0\_0\_, dietpatter0\_.food\_item\_id as food\_ite8\_0\_0\_, dietpatter0\_.hps\_user as hps\_user4\_0\_0\_, dietpatter0\_.last\_updated as last\_upd5\_0\_0\_, dietpatter0\_.quantity as quantity6\_0\_0\_, foodinterv1\_.display\_order as display\_2\_3\_1\_, foodinterv1\_.type\_name as type\_nam3\_3\_1\_, foodinterv1\_.type\_name\_hin as type\_nam4\_3\_1\_, fooditem2\_.calorie as calorie2\_4\_2\_, fooditem2\_.carbs as carbs3\_4\_2\_, fooditem2\_.fat as fat4\_4\_2\_, fooditem2\_.fibre as fibre5\_4\_2\_, fooditem2\_.food\_category\_id as food\_ca13\_4\_2\_, fooditem2\_.kapha as kapha6\_4\_2\_, fooditem2\_.pitta as pitta7\_4\_2\_, fooditem2\_.proteins as proteins8\_4\_2\_, fooditem2\_.quantity as quantity9\_4\_2\_, fooditem2\_.serving as serving10\_4\_2\_, fooditem2\_.status as status11\_4\_2\_, fooditem2\_.vatta as vatta12\_4\_2\_, locale\_foo3\_.alias as alias2\_7\_3\_, locale\_foo3\_.food\_item\_id as food\_ite8\_7\_3\_, locale\_foo3\_.kapha\_comment as kapha\_co3\_7\_3\_, locale\_foo3\_.locale as locale4\_7\_3\_, locale\_foo3\_.name as name5\_7\_3\_, locale\_foo3\_.pitta\_comment as pitta\_co6\_7\_3\_, locale\_foo3\_.vata\_comment as vata\_com7\_7\_3\_, locale\_foo3\_.food\_item\_id as food\_ite8\_7\_0\_\_, locale\_foo3\_.id as id1\_7\_0\_\_, localefood5\_.food\_category\_id as food\_cat4\_6\_5\_, localefood5\_.locale as locale2\_6\_5\_, localefood5\_.name as name3\_6\_5\_, localefood5\_.food\_category\_id as food\_cat4\_6\_1\_\_, localefood5\_.id as id1\_6\_1\_\_ from diet.diet\_pattern dietpatter0\_ inner join diet.food\_interval foodinterv1\_ on dietpatter0\_.food\_interval\_id=foodinterv1\_.id inner join diet.food\_item fooditem2\_ on dietpatter0\_.food\_item\_id=fooditem2\_.id inner join diet.locale\_food\_item locale\_foo3\_ on fooditem2\_.id=locale\_foo3\_.food\_item\_id inner join diet.food\_category foodcatego4\_ on fooditem2\_.food\_category\_id=foodcatego4\_.id inner join diet.locale\_food\_category localefood5\_ on foodcatego4\_.id=localefood5\_.food\_category\_id where localefood5\_.locale=? and locale\_foo3\_.locale=? and dietpatter0\_.hps\_user=? and dietpatter0\_.created\_on\_date=? This query has five inner joins so might take a lot of execution time if server was being used by many users at the same time. Also the query is being used repeatedly.
8. There is no consistency in defining tables in models across hpsserver,mudra and corporate apps.
9. There is additional empty initialisation in dietIntervalReport and dietReport dto.
10. Different code structuring across hpsserver,mudra and corporate apps.
11. Direct query execution call in controller used for mudra app making it susceptible for SQL injections and gives access to dangerous actions like deleteAll.
12. Inconsistency in data for finding out pheno type overall info. For VK its Your response to questionaire tells rest all its Your constitution is there should be consistency across the data being displayed to user.
13. No checks in place to avoid caller changing the value.
14. Null check missing in the code.
15. Name footItemName is used in place of foodItemName in dietPreferenceServiceImpl.
16. Input validations are missing.
17. No authentication in place for admin console api written in python so its possible that security can be compromised.
18. Different set of api written for both mobile version and web version for goals model.
19. Some endpoints in resources has no permissions attached to the end points while resources in mobile do.
20. Commented code which is not being used still exists in production.Remove the commented code.
21. In health\_report\_language\_converter no code written for Marathi language also the if else conditions could be optimized.
22. Follow consistent structure while executing if else statements for all languages in questionnaire\_help\_text\_language\_converter as Marathi is inferred from json rest are not.
23. No code written for get\_page\_data in Hindi language.
24. No need of using created\_ts and created\_at both date-time fields in a single model for region. Same is the case for updated\_at and updated\_ts.

Mitigations:

1. There are three answers for this first log the occasions that are going on to the database second ensure the document is put away in a catalog which isn't accessible to all clients in home index by renaming it to an option that is other than nohup.out which is the thing that the test direction usually is when utilizing nohup. third compose a content which physically stacks the logged record in database every day on the off chance that you would like to keep up a sign on regular schedule and, at that point erase the document from server as it will keep on expanding with more exercises on server and consequently expanding your server extra room.
2. Used named query instead of the regular query using string builder.
3. Apply authentication while accessing any URL apart from home page of app.
4. The credentials should not be exposed in the server so as to avoid exploiting of data by unprecedented sources.
5. Date field should be formatted properly before passing it to query.
6. Number of steps in steps taken data should be sum of steps in steps history.
7. The query should be scripted as a view and should have user id,created date and locale as parameters to optimize and secure the query.
8. Have a consistent structure for naming tables across all modules in the app.
9. Remove empty initialization.
10. Code structure should be same across all apps. I.e. the code flow should be the same in all apps.
11. Execution of query should never happen in controller it should happen in serviceImpl.
12. Data should be consistent across all responses when determining the type of pheno.
13. Check the user authentication while calling the api either using role based or permission based.
14. Add null checks to all fetched data to avoid data display errors.
15. Name convention should be followed to give meaningful names while coding.
16. All inputs should be validated before inserting in database.
17. Authentication check must be in place for executing endpoints of api.
18. There should be minimum code duplication and all end points should be used from single model instead of having different models.
19. All end points should have permission checks in place to prevent them for being misused in terms of security.
20. Remove redundant code from production code.
21. Code written for Marathi language in health report language converter.
22. Code consistency should be maintained in questionnaire\_help\_text\_language\_converter and if else conditions need to be optimized.
23. Write code to get page data in Hindi language.
24. A single field of created\_at and updated\_at will be sufficient for logging the created and updated time stamp. Time can be extracted from date-time if its required.