Project Part 4: Architectural Design Principles, Usable Security Principles, Protection Poker and Bug Fixes.

**Group Members**

Rushi Bhatt (rbhatt)

Neetish Pathak (npathak2)

Bhavya Bansal (bbansal)

Aswin Anil Kumar (aanilku)

Architectural Design Violations:

**1. Architectural Design Violated: Complete Mediation**

A regular user can log in with his/her credentials to a user account and then open the admin url to get access to admin dashboard with full admin privileges. Not reauthorizing when navigating to admin page. Authorization done only at login.

#### **Test Case:**

##### Login as a normal user without administrative privileges

##### Copy and paste the below URL:

##### http://localhost:8081/openmrs-standalone/admin/

**Expected Result:** User shall not be able to access this page as he/she has not been granted administrative privileges.

##### **Actual Result:** User was able to access the page and was allowed to perform update/ delete operations which is in direct violations with the access control policy.

Complete mediation is not followed. Authorization happens only when the user logs into his/her account. While navigating to the admin dashboard, authorization check should again happen.

**How can we fix this:** To fix this the system should do an authorization check each time the admin is loaded and not only at the time of user authentication.

**2. Architectural Design Violated: Detect Intrusion**

We are able to launch attacks on the system with tools like OWASP ZAP. The system does not lock when its is flooded with requests. No intrusion prevention mechanisms in place.

#### **Test Case:**

1. Configure OWASP ZAP
2. Open OWASP and from the top right corner, select attack mode
3. Log into OpenMRS as admin and copy the url
4. In the ZAP tool, Paste the URL in “URL to attack” field and click the Attack button
5. This will send around 150 requests to the app in under 5 seconds

##### **Expected Result:** The app should detect the unusually high volume of requests coming from the same address and block further requests.

##### **Actual Result:** We are able to automatically probe the website for vulnerabilities. ZAP was able to find 4 vulnerabilities with this attack alone.

**How can we fix this:**

The system should be intelligent enough to recognize probing/scanning attacks and block further requests from the malicious IP address.

We should make the system intelligent enough to detect probing/malicious attacks. Some of the characteristics of such attacks are:

1. All the requests will be from the same IP address
2. All these requests happen in an unusually short period of time.

The system should watch out for this pattern of request and when detected, block further request from the address for a period of time.

**3. Architectural Design Violated: Quiet Your Error Messages**

Attackers can easily get access to error stack traces on the frontend by giving invalid inputs. Instead of failing safely by showing an error page, the stack trace is exposed which contains all file names, class names and method names.

#### **Test Case:**

##### Log in as admin

1. Open “Find Patient Record”
2. Open any of the patients
3. Note the url, it will have a patientID parameter.

Now change the patientID to some random string(invalid input):

<http://localhost:8081/openmrs-standalone/coreapps/clinicianfacing/patient.page?patientId=ABCD>

You will get an error page with a NullPointerException stack trace that exposes a lot of class names and library names. Keep refreshing the page with the same or different random string and you will get the same page.

##### **Expected Result**: The system should fail safely by displaying an error message.

##### **Actual Result:** Exposing a stack trace to a user is a major security risk as an attacker can get a good idea of libraries and technology stack used by the web application. This stacktrace is also getting logged in the server. This can cause memory overflow issues. In addition to that the app is not doing anything to prevent repeated attacks.

**How we can fix this:**

Exceptions should be handled properly at the backend and the stack trace should not be propogated to front end. When an exception occurs, send a meaningful message to the front end and not let the user know the internal details of what excatly happened.

**2. Usable Security Principles**

Three examples of usable security violations in openMRS are:

1. Expected ability:

openMRS violates principle of expected ability as their is a section where a health care professional can mark a patient deceased, by opening a web page which ask for cause of death. The feature is provided as a dropdown but there are no values displayed in dropdown and nothing is shown which can give an idea to the user about the issue. This clearly violates the given principle.

**Test Case:**

* Login as admin
* Click on “find patient record”
* Search and choose a patient
* Click on “Mark Patient Deceased” option on the right hand side
* Click on checkbox provided
* Enter date of death and click on dropdown for “Cause of Death”

**Expected Results:**

System shall show default values configured for “cause of death” and if not then shall show a message to user indicating that default values need to be configured.

**Actual Results:**

System does not show any default values, and no message is displayed to user either which can help user to understand the issue and fix it. Moreover, since this value is mandatory to mark patient as deceased, user can not proceed ahead until and unless he/she makes a selection.

2) Clarity:

openMRS also violates the principle of clarity at many places in the web application. One such example where it violates this principle is the action related to deleting a user of openMRS. If an admin clicks on “Delete User” button, then system does not ask admin confirmation nor does show what will happen as a result of this critical action.

**Test Cases:**

* Login as admin
* Click on “System Administration”
* Click on “Advanced Administration”
* Click on “Manage Users”
* Search any user in the system
* User profile will be displayed, click on “Delete User” button

**Expected Results:**

System shall show a confirmation dialog with details of what will happen as a consequence of this action.

**Actual Results:**

System does not ask for confirmation and directly deletes the selected user from the system.

2) Identifiability:

openMRS violates the principle of Identifiability in the admin console for many functionalities. One such place is when an admin has to add a new user. A new user can be added at two places, one is using System Administration -> Advanced Administration -> Manage Users -> Add User and the other one is System Administration -> Manage Accounts -> Add New Account.

**Test Cases:**

* Login as admin
* Click on “System Administration”
* Click on “Advanced Administration”
* Click on “Manage Users”
* Click on “Add User”, you can add a new user by filling details here and save
* Now go back to home page
* Click on “System Administration”
* Click on “Manage Accounts”
* Click on “Add New Account”

**Expected Results:**

Both interface shall have same set of values in the form for a new user as the functionality expected is same.

**Actual Results:**

Adding a user via “Manage users” path, has an extra feature to provide security questions available through “advance options” on the form but the same is absent in the form available via “Add New Account” link.

**3. Protection Poker**

Functional requirements for OpenMRS:

1. **Patient portal:** OpenMRS system so far only supports medical professionals and administrators and doesn’t provide any functionality for patients. However, we can add the patient login functionality where patients can perform some basic operations like

* view their medical records
* View their appointment and prescription history
* request an appointment

Based on the feedback of these features, we can further include more advanced features in the patient portal.

**Database tables affected:** OpenMRS already stores all the patient information in the database. We just need to add credentials for the patients through which they can access the openMRS portal. So if we add a role in the “user\_role” table to differentiate the users and patients, we can use the same database tables for storing information.

So we will require newly created role: ‘Patient’ and that will in turn change the data of the following tables:

**user\_role, role, role\_privilege, role\_role** with newly added entry for patient role. But it won't change the database schema.

1. **Two factor authentication:** OpenMRS authenticates users by simple username and passwords, which is not the safest way to perform authentication, considering the amount of PHI data the system can expose if goes in the wrong hand. We can add the two factor authentication using the additional security questions or images in order to add one extra layer of protection to make the system secure.

**Database tables affected:** Based on how we try to implement the two factor authentication, we need to add different things in the database.

* For simple implementation with security question, we can add security question and answer while creating the user account. However, these fields are already created in the users table.
* To implement it using Phone number, we just need the contact information of the user, so we can add that in the person table.
* **phone VARCHAR(30) in “person” table**

1. **Forgot password feature:** If the admin or user forgets the username or password for the OpenMRS, there is no way to retrieve the information and validate that person because the OpenMRS system doesn’t require the use of email addresses for the creation of new user. However, if we ask for the valid email address for a person while creating an account, we can sync that email account with the username, and can provide the user with the “Forgot password” feature that will in turn send the username and password information to email address provided by the user.

**Database tables affected:** we just need to add email address field with each user. Since the “user” table is associated with person through pid, we can add email in the person table.

* **email VARCHAR(30) in “person” table**

1. **Auto-sync with fitbit:** OpenMRS let’s you manually add various vital information to the patient's health records, but with modern gadgets like fitbits, we can keep track of every activity and vital information of the patient with utmost accuracy if we provide the auto-sync feature of OpenMRS with such devices.

**Database tables affected:** we just need to store the fitbit account credentials for each patient. Afterwards, we can use that credentials to get the data from fitbit API’s and directly use that in the system.

* **fitbit\_id VARCHAR(30) in “patient” table**
* **fitbit\_enabled BOOLEAN in “patient” table**
* **fitbit\_password VARCHAR(128) in “patient” table**

1. **Integrating recommendations:** Once we create the patient portal, we can integrate a recommendation system that will show various suggestions to patients based on their past medical history and prescriptions. These recommendations may include various qualified doctors specialising in particular category, medicines and nearby pharmacies. Patient can use these information to make better and informed decisions.

**Database tables affected:** We can use microsoft azure recommendation service to get the recommendations and store it in the database. Based on what all are we recommending the patients, we can modify the database. However, we will have to create a new table called “recommendations” for that.

**Table: recommendations**

* **patient id VARCHAR(30)**
* **recommendation\_type VARCHAR(20) -** doctor, medicine or pharmacy
* **name VARCHAR(50)**
* **details VARCHAR(200)**  - This details can be changed based on recommendation type

For exp., for pharmacy recommendations, we can show address from the nearby location of patient address.

Protection poker: 1,2,3,5,8,13,20,40,100

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Feature | Ease Points | Value Points | Security Risk | Ranking |
| Patient Portal | 20 | 20 | 400 | 4 |
| Two factor authentication | 13 | 100 | 1300 | 1 |
| Forgot password feature | 8 | 100 | 800 | 2 |
| Auto-sync with fitbit | 40 | 13 | 520 | 3 |
| Integrating recommendations | 13 | 8 | 104 | 5 |

Ease points: Among the 5 features, the hardest feature for the attackers to exploit is Forgot password feature. The attacker will have to get access of the email id and password of the user in order to retrieve/change the current password. Compared to that, two factor authentication can be broken easily by social engineering attack, where attacker can try to get the security questions right. Same is the case for Integrating recommendation where the attacker need to get access of the patient portal and therefore bypass the two-factor authentication. The easiest feature for the malicious users to crack would be auto-sync with fitbit. Here, attackers just need to steal the fitbit device and can potentially change the vital information of the patient is the application is logged in.

Value Points: Among the 5 features, the highest impact of the breach would be for Two-factor authentication and Forgot password feature as both of these would directly give access of the admin/user account to the malicious user, who, with admin privileges, can perform any action he/she wants. Since Patients have limited privileges, the patient portal feature will not have as much impact as the admin portal breach.Finally, the feature auto-sync with fitbit and integration recommendations have low impact value as they can have limited capabilities and access over the patient records.

Based in the ease and value points assigned using the protection pocker, we can say that the features with ranking risks are sorted below:

1. Two-factor authentication
2. Forgot password
3. Auth-sync with fitbit
4. Patient portal
5. Integrating recommendation

# 

# 

# 

# 4. Bug Fixes

|  |  |
| --- | --- |
| Bug Fix 1 | SQL Injection in MigrateAllergiesChangeSet.java |
| Code Difference | https://github.com/NeetishPathak/openmrs-core/commit/0c31fc57a84abbb41cfe10e37a6d169f5dc49baa |
| Code correctness | The SQL query is handled using a preparedStatement which is a mitigation technique as learnt from [OWASP top 10 lecture](https://dab0b0c2-a-bf11e6be-s-sites.googlegroups.com/a/ncsu.edu/csc515-software-security/Schedule-of-Subjects/Prelim/2017.2b%20OWASP%20Top%2010%2C%20part%201.pdf?attachauth=ANoY7criRmAAldZjtY4HcEvmu3LPDJsOXA7U3SrlfmxTKUHZHGKlZPYRTFBpUvWq49ciOD0qD-jpUUiCYFxG0D64A2s9g_1-PhN8S4oO5p99PrFmJ9A5Xg0JyiMXi0zkjnOxUx324L629njSomwtJEXREKPzpkl2OZKqxc0R7Cu9pT4GTJ4srMTw0PC_m08TR37lATVL8JiJB3K8WX7VSc9wxWLym_cuomBfgJQYRbZF88i8jvw5U870dlPiDh14QaXEE8flcQC-QuRwZsxIuWxZlUweYlcpxV-HZy0usJixI2mJSlaDOX0%3D&attredirects=0)  Here, rather than passing the uuid in the sql query, we use the PreparedStatement object pStmt to prepare the query and set the uuid parameter. |
| Reference | Page 35 of [Report](https://drive.google.com/a/ncsu.edu/file/d/0B7qn0T6hM7hsUXVLNHdhblIycmM/view?usp=sharing)  This is listed as OpenMRS deficiency 1 in our Report 3 |

|  |  |
| --- | --- |
| Bug Fix 2 | Preventing Log Forging Attack |
| Code Difference | https://github.com/NeetishPathak/openmrs-core/commit/e3d6cd0bd281d5d8e49d0ac26a6870ad838d0928 |
| Code correctness | The fix will check the user input and will only perform next set of actions if the user input is validated and pass the validation check. If the validation fails, then system will simply log the incorrect attempt with appropriate logs. |
| Reference | Page 6 of [Report](https://drive.google.com/a/ncsu.edu/file/d/0B7qn0T6hM7hsUXVLNHdhblIycmM/view?usp=sharing)  This is listed as OpenMRS deficiency 5 in our Report 3 |

|  |  |
| --- | --- |
| Bug Fix 3 | Moving hardcoded password to property file |
| Code Difference | https://github.com/NeetishPathak/openmrs-core/commit/2fb2a441dbf7cc803c5be9071bbfbefd2802d07b |
| Code correctness | The password was originally hard coded in the above java file. We removed the password to a property file and added code in a static block to fetch the password from the log file. When this file is loaded for the first time, static block will get executed and fetch the password. |
| Reference | Page 13 of [Report](https://drive.google.com/a/ncsu.edu/file/d/0B7qn0T6hM7hsUXVLNHdhblIycmM/view?usp=sharing)  This is listed as OpenMRS deficiency 6 in our Report 3 |

|  |  |
| --- | --- |
| Bug Fix 4 | Dynamic code evaluation: Unsafe deserialization. Handled using look-ahead deserialization pattern using Apache Commons IO (org.apache.commons.io.serialization.ValidatingObjectInputStream) |
| Code Difference | Screen Shot 2017-11-14 at 7.18.47 PM.png  Screen Shot 2017-11-14 at 7.18.17 PM.png  https://github.com/NeetishPathak/openmrs-core/commit/75691d1e1fdb987fbc5fef6af508883aca1e9bbb?diff=unified |
| Code correctness | Deserializing user-controlled object streams at runtime can allow attackers to execute arbitrary code on the server, abuse application logic, and/or lead to denial of service. Here, we ensure not to deserialize untrusted data without validating the contents of the object stream by use look-ahead deserialization pattern with one of its existing implementations like Apache Commons IO.  The correct way to implement a deserialization is to use a safeInputStream (eg. ValidatingObjectInputStream) and test for valid class name as referrred from <https://adityagollapudi.wordpress.com/2016/05/22/unsafe-java-deserialization/>  So, in the change made, there is a check for the valid class name using ois.accept(“Person”) before deserializsation takes place. If the class name does not match ClassNotFoundException is thrown |
| Reference | Page 29 of [Report](https://drive.google.com/a/ncsu.edu/file/d/0B7qn0T6hM7hsUXVLNHdhblIycmM/view?usp=sharing) JavaSerializationTest.java, line 48 (Dynamic Code Evaluation: Unsafe Deserialization)  Mentioned as OpenMRS deficiency # 9 in report 3 |

|  |  |
| --- | --- |
| Bug Fix 5 | Fix for Server-side request forgery. Preventing attacks on a network connection to a third-party system using user-controlled data for resource URI |
| Code Difference | Screen Shot 2017-11-14 at 8.25.09 PM.png  ...  Screen Shot 2017-11-14 at 8.01.52 PM.png |
| Code correctness | We use an example list of the valid locations before initiating a connection to a third-party system using user-controlled data. In the fortify static code analysis, it was mentioned that *if user data is necessary to build the destination URI, use a level of indirection: create a list of legitimate resource names that a user is allowed to specify, and only allow the user to select from the list.* Here we have validated that the user input is from valid locations that are expected. We could not provide an option in GUI to altogether limit the user input to valid locations since that fix is more involved (making changes to GUI). We provide a toy example of whitelisting here.  The input location is tested against the whitelist using validLocs.contains(“loc”). An IllegalArgumentException is thrown if the location is not valid.  https://github.com/NeetishPathak/openmrs-core/commit/1e1d7b4f36896313f1ffce14e0a9daea875d8f13?diff=unified#diff-9ce91be2e2cbad1fe680bebfa346d4ba |
| Reference | Page 31 of [Report](https://drive.google.com/a/ncsu.edu/file/d/0B7qn0T6hM7hsUXVLNHdhblIycmM/view?usp=sharing) Server-Side Request Forgery  Mentioned as OpenMRS Deficiency 10 in report 3 |

Link for all the changes from the base repo:

<https://github.com/openmrs/openmrs-core/compare/master...NeetishPathak:master#diff-9ce91be2e2cbad1fe680bebfa346d4ba>