Architectural Design Principle, Usability Security Principles, Protection Poker, Bug Fixes

Project Phase IV

Ananthram Eklaspuram Lakshmanasamy - aeklasp
Arun Jaganathan - ajagana
Bhaskar Sinha - bsinha
Nivedita Natarajan – nnatara2

1. Architectural Design Principles:

Note: In the following test cases, the naming convention followed is: DV_XX_1 where DV - Design Violation; XX - Abbreviation of the principle

1. Fail Securely

The OpenMRS system violates the design principle of Fail Securely. Fail securely is the concept implemented to ensure security when the application fails. In OpenMRS, when trying to navigate to a page not found in the system or broken links in the application, instead of throwing a standard 404 not found page, it displays a stack trace encountered by tomcat that caused the page to not display. This involves a lot of information (such as tomcat version number) that can be used by a potential malicious user to break into the system.

Test Case ID: DV_FS_1

Test Case Description: DV_FS_1 tests the OpenMRS system to check if it fails securely or invovles information during a failure.

Instructions to Execute DV_FS_1:

1. Launch OpenMRS application in the local machine.
2. Once the application starts, navigate to the following page: **http://localhost:[port]/openmrs-standalone/referenceapplication/invalid.page**
3. Replace [PORT] with the port the OpenMRS is running. By default it is 8081

Results:

Test Case ID	Expected Results	Actual Results	Test Case Results
DV_FS_1	A standard 404 not found page is loaded	A stack trace is displayed by tomcat which also includes its version number	FAIL - System fails insecurely.

2. <u>Principle of Least Privileges:</u>

The OpenMRS system violates the principles of least privileges. The '*Privilege Level: Full'* role gives access to all the APIs in the system. Almost all roles in the system hold full privileges to do the tasks that require minimal privileges. This violatest the principle of least privileges.

Test Case ID: DV_LP_1

Test Case Description: DV_LP_1 tests the OpenMRS system if there are roles with unncessary high privilege.

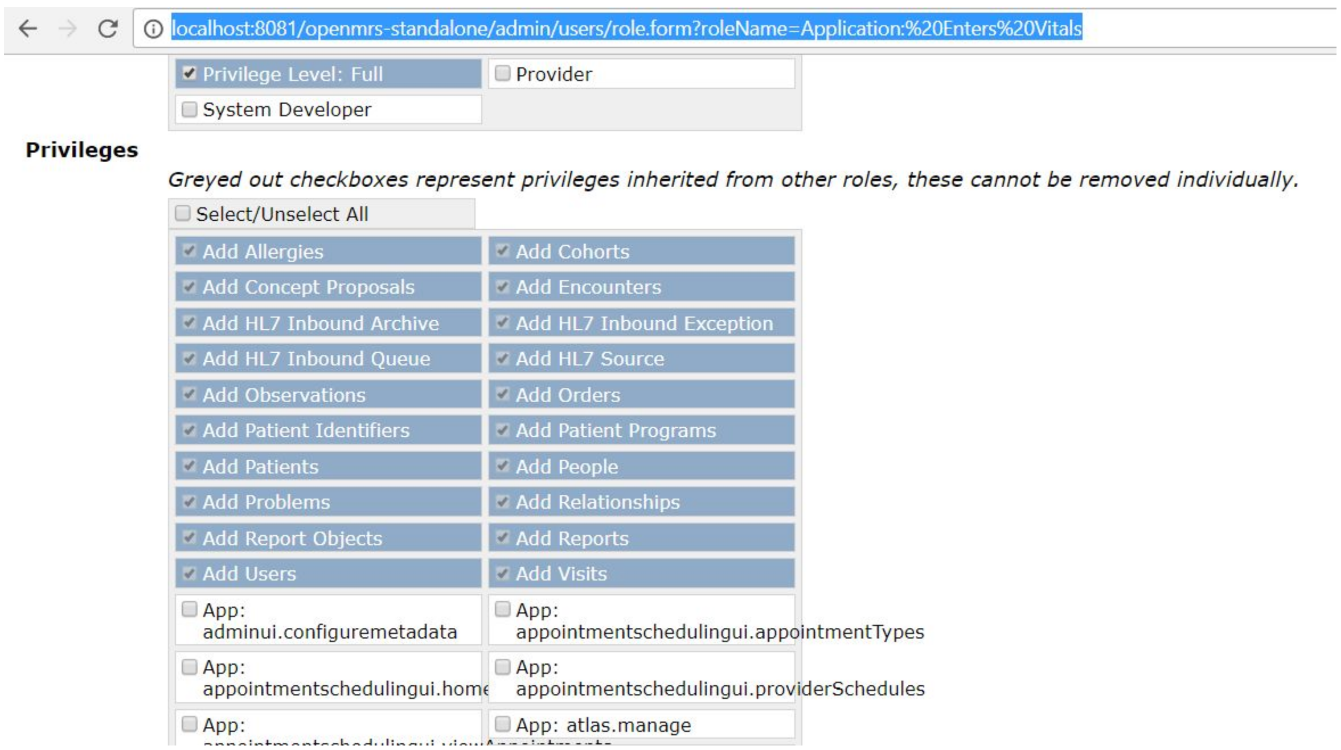
Instructions to Execute DV_LP_1:

1. Launch OpenMRS application in the local machine and login as Admin with username: admin and password: Admin123.
2. Navigate to the following page: **http://localhost:[port]/openmrs-standalone/admin/users/role.list**
3. Replace [PORT] with the port the OpenMRS is running. By default it is 8081
4. The above link shows a list of roles and mentions the Privilege level of each roles. We can see that many roles like, '*Application: Enter Vitals*', '*Application: Record Allergiers*', etc, inherits '*Privilege Level: Full*'.
5. Click on '*Application: Enter Vitals'* and the page will navigate to: http://localhost:8081/openmrs-standalone/admin/users/role.form?roleName=Application:%20Enters%20Vitals
6. Let us examine the privileges this Role has.

Results:

Test Case ID	**Expected Results**	**Actual Results**	**Test Case Results**
DV_LP_1	'*Application: Enter Vitals'* should not hold any extra privileges. For ex, this role should not hold user administration privileges	'*Application: Enter Vitals'* has API access to 'Add Users' and 'Add People'	FAIL - User administration does not follow the principle of least privilege as for the role to record allergies to holds all privileges.

Screenshot:



localhost:8081/openmrs-standalone/admin/users/role.form?roleName=Application:%20Enters%20Vitals

✓ Privilege Level: Full	Provider
System Developer	

Privileges

Greyed out checkboxes represent privileges inherited from other roles, these cannot be removed individually.

Select/Unselect All

✓ Add Allergies	✓ Add Cohorts
✓ Add Concept Proposals	✓ Add Encounters
✓ Add HL7 Inbound Archive	✓ Add HL7 Inbound Exception
✓ Add HL7 Inbound Queue	✓ Add HL7 Source
✓ Add Observations	✓ Add Orders
✓ Add Patient Identifiers	✓ Add Patient Programs
✓ Add Patients	✓ Add People
✓ Add Problems	✓ Add Relationships
✓ Add Report Objects	✓ Add Reports
✓ Add Users	✓ Add Visits
App: adminui.configuremetadata	App: appointmentschedulingui.appointmentTypes
App: appointmentschedulingui.home	App: appointmentschedulingui.providerSchedules
App:	App: atlas.manage

The above is the screenshot of Application: Enter Vitals Role (as you can see from URL). We can see that ‘Add users’ privileges has been granted to this role, which is a violation of this principle.

3. Do Not Allow Modifications Without A Trace:

In order to follow the property of non-repudiation, any actions taken in the system should be logged. For instance, An employee can maliciously modifies patient information for personal agenda. While it may not be possible to deter the attacker, if proper logging is followed and monitored by an administrator, it is still possible to revert this action. OpenMRS violates this principle as it doesn’t provide non-repudiation for a lot of actions in the system.

Test Case ID: DV_MT_1 (MT stands for Modification Trace)

Test Case Description: DV_MT_1 tests the OpenMRS system to check if the system logs which user performs the action when editing patient details.

Instructions to Execute DV_MT_1:

1. Launch OpenMRS application in the local machine and login as Admin with username: admin and password: Admin123.
2. Once the application starts, navigate to the following page: **http://localhost:[PORT]/openmrs-standalone/coreapps/findpatient/findPatient.page?app=coreapps.findPatient**
3. Replace [PORT] with the port the OpenMRS is running. By default it is 8081
4. The above URL points to 'Find Patient' page. Search for '10017E' which is the patient details of Thomas Smith.
5. Edit any information of this patient - say, the DOB of the patient and save the patient detials now.
6. Open the log file present in [openMRS standalone root folder]/appdata/openmrs.log
7. Scroll down to the bottom of the log to check for what has been logged.

Results:

Test Case ID	**Expected Results**	**Actual Results**	**Test Case Results**
DV_MT_1	Type of Modification: EDIT Who: System Administrator/ Super User When: Date/Time of access What: Action invoked (Save Patient)	INFO - LoggingAdvice.invoke(115) |2017-11-12 16:28:18,271| In method PatientService.savePatient. Arguments: Patient=Patient#44, INFO - LoggingAdvice.invoke(155) |2017-11-12 16:28:18,308| Exiting method savePatient	Fail - Who performed the action isn't logged

2.Usable Security Principles:

2.1. Appropriate Boundaries (Principle 2)

OpenMRS violates the principle of Appropriate boundaries. According to the principle, the Interface (App control logic) should distinguish objects and actions along boundaries (user) that relate to important issues such as “need to know” or “least privilege”. Granularity of user roles is too broad, so each role may have more authority than intended. For instance , in OpenMRS doctors and nurses have access to admin console page by using the URL.

Test Case

Test Case ID	US_2.1
Description	Poor privilege boundaries between Doctors, nurses and Admin
Steps	1. Open OpenMRS, get to login page, login as user Doctor using following credentials: Username: Doctor Password: Doctor123 2. Use the following admin console page URL in the same browser: http://localhost:8080/openmrs-standalone/admin/index.htm. 3. Log Out of the system and repeat the steps 1 and 2 with user Nurse, with following credentials: Username: nurse Password: Nurse123
Expected Results	Doctor and nurse users should not be able to access admin page
Actual Results	Doctor and nurse users are able to access admin page
Test Case Result	FAIL

2.2. Trusted Path (Principle 7)

OpenMRS violates the principle of Trusted Path. According to the principle of Trusted path, The interface must provide an unspoofable and faithful communication channel between the user and any entity trusted to manipulate authorities on the user’s behalf. This makes the system vulnerable to CSRF attacks.

Test Case

Test Case ID	US_2.2
Description	System vulnerable to CSRF attacks
Steps	1. Please replace the values of the following variables according to your OpenMRS set up [PORT] - Port on which OpenMRS is running [PATIENT] value for patientId found in step 9. [ALLERGY] value for allergyId found in step 11. 2. Launch the openmrs standalone application in the Chrome Browser and bookmark the page. 3. Now open the bookmark manager in Chrome (Ctrl+Shift+O on Windows/Command+option+B on Mac.). 4. Right click the recently bookmarked openmrs application and click on Edit. 5. Set the URL value as: *javascript:var my_params=prompt("Enter your parameters",""); var Target_LINK=prompt("Enter destination", location.href); function post(path, params) { var xForm= document.createElement("form"); xForm.setAttribute("method", "post"); xForm.setAttribute("action", path); for(var key in params) { if(params.hasOwnProperty(key)) { var hiddenField = document.createElement("input"); hiddenField.setAttribute("name", key); hiddenField.setAttribute("value", params[key]); xForm.appendChild(hiddenField); } } document.body.appendChild(xForm); xForm.submit(); } parsed_params={}; my_params.split("&").forEach(function(item) {var s = item.split("="), k=s[0], v=s[1]; parsed_params[k] = v;}); post(Target_LINK, parsed_params); void(0);* 6. Save the bookmark. 7. Log in as the Admin user to the system, create a new patient by clicking Register a patient. 8. Now on the patient page, click on the allergies link to open up the allergies page. 9. Make a note of “patientId” parameter’s value in the URL. 10. Click on “Add New Allergy” and select “Aspirin” in the Drug Tab and click on “Save”. 11. You will now be in the Allergies page. Click on the Edit action in the Actions column for the Allergy you created. Note the “allergyId” parameter’s value in the URL. Click on “Cancel” to go back to the allergies page. 12. Open the bookmark you previously saved. 13. You’ll get a prompt asking for parameters. 14. Supply the following value: *patientId=[PATIENT]&returnUrl=/openmrsstandalone/coreapps/clinicianfacing/patient.page?patientId=[PATIENT]&allergyId=[ALLERGY]&action=removeAllergy* 15. After submitting the previous data, you’ll be prompted for destination, Supply the following value to it:

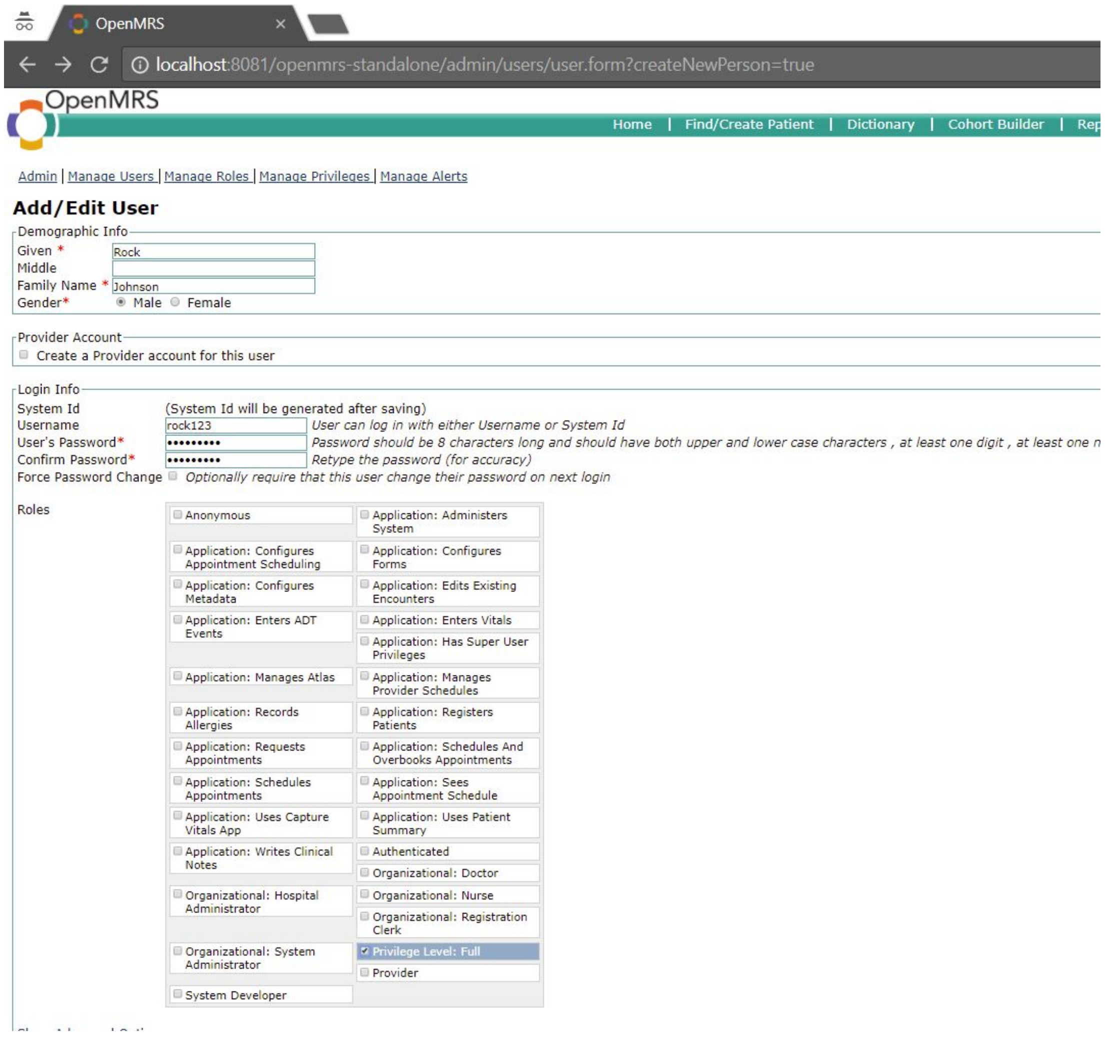
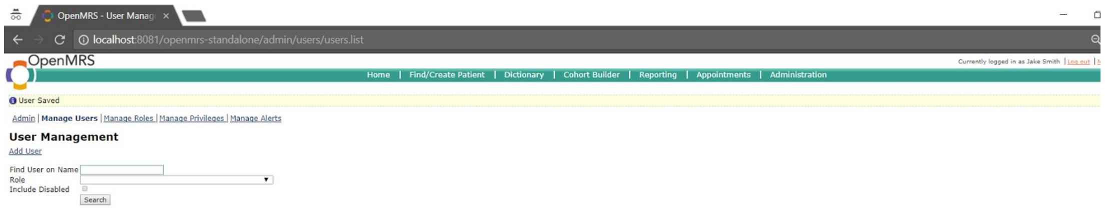
	http://localhost:[PORT]/openmrsstandalone/allergyui/allergies.page?patientId=111&
Expected Results	There should be no change to the patient allergy data
Actual Results	The allergy which is linked to [ALLERGY] is removed for the patient [PATIENT]
Test Case Result	FAIL We can deduce that the system is insecure.

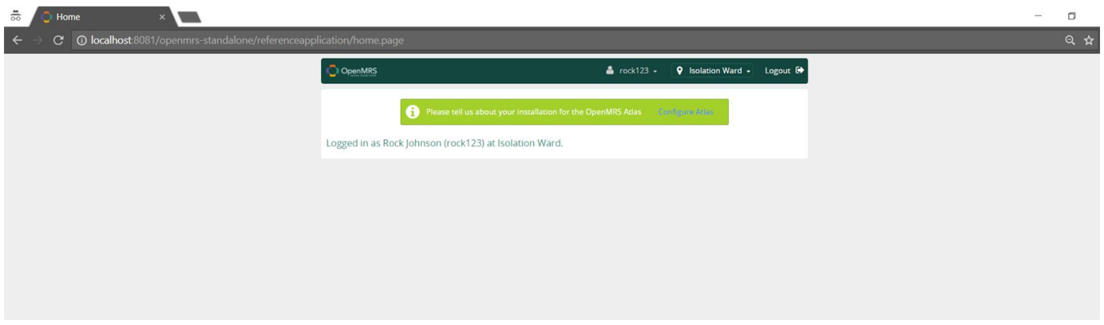
2.3. Explicit authorization (Principle 3)

OpenMRS violates the principle of Explicit authorization. According to the principle. a user's authorities must only be provided to other actors as a result of an explicit action that is understood by the user to imply granting. The application should explicitly authorize the user's action when there is a escalation of privilege. OpenMRS does not efficiently check for user authorization while adding a system user, this can be easily exploited to carry out malicious activities especially in insider threat scenarios.

Test Case

Test Case ID	US_2.3
Description	Doctor user can add and remove users by using the Admin console URL (http://localhost:8081/openmrs-standalone/admin/index.htm)
Steps	1. Open OpenMRS application 2. Log in as the Doctor user (default user) using the credentials Username: doctor and Password: Doctor123. 3. Use the following URL http://localhost:8081/openmrs-standalone/admin/index.htm 4. Click on Manage users under Users tab. 5. Click on Add user 6. Click Next under the Create a new person tab. 7. Enter a Given Name(Example: Rock) and Family Name(Example: Johnson), choose gender (Example: Male), Enter a password(Example: Doctor123), enter a Username(Example: rock123). Select the role “Privillege Level: Full”. 8. Save the user. 9. Now try to login ass that user.

Expected Results	OpenMRS should not allow to create a user in the first place and also not allow to log in as the newly created user
Actual Results	User is created and aslo allowed to login as that user.  OpenMRS localhost:8081/openmrs-standalone/admin/users/user.form?createNewPerson=true OpenMRS Home | Find/Create Patient | Dictionary | Cohort Builder | Rep Admin | Manage Users | Manage Roles | Manage Privileges | Manage Alerts **Add/Edit User** Demographic Info Given * Rock Middle Family Name * Johnson Gender* [✓ Male] Female Provider Account ☐ Create a Provider account for this user Login Info System Id (System Id will be generated after saving) Username rock123 *User can log in with either Username or System Id* User's Password* ••••••••• *Password should be 8 characters long and should have both upper and lower case characters , at least one digit , at least one n* Confirm Password* ••••••••• *Retype the password (for accuracy)* Force Password Change ☐ *Optionally require that this user change their password on next login* Roles ☐ Anonymous ☐ Application: Administers System ☐ Application: Configures Appointment Scheduling ☐ Application: Configures Forms ☐ Application: Configures Metadata ☐ Application: Edits Existing Encounters ☐ Application: Enters ADT Events ☐ Application: Enters Vitals ☐ Application: Has Super User Privileges ☐ Application: Manages Atlas ☐ Application: Manages Provider Schedules ☐ Application: Records Allergies ☐ Application: Registers Patients ☐ Application: Requests Appointments ☐ Application: Schedules And Overbooks Appointments ☐ Application: Schedules Appointments ☐ Application: Sees Appointment Schedule ☐ Application: Uses Capture Vitals App ☐ Application: Uses Patient Summary ☐ Application: Writes Clinical Notes ☐ Authenticated ☐ Organizational: Doctor ☐ Organizational: Hospital Administrator ☐ Organizational: Nurse ☐ Organizational: Registration Clerk ☐ Organizational: System Administrator ☑ Privilege Level: Full ☐ Provider ☐ System Developer  OpenMRS - User Manag localhost:8081/openmrs-standalone/admin/users/users.list OpenMRS Currently logged in as Jake Smith | Log out Home | Find/Create Patient | Dictionary | Cohort Builder | Reporting | Appointments | Administration User Saved Admin | **Manage Users** | Manage Roles | Manage Privileges | Manage Alerts **User Management** Add User Find User on Name Role Include Disabled ☐ Search

	 Home localhost:8081/openmrs-standalone/referenceapplication/home.page OpenMRS rock123 Isolation Ward Logout Please tell us about your installation for the OpenMRS Atlas Configure Atlas Logged in as Rock Johnson (rock123) at Isolation Ward.
Test Case Result	FAIL OpenMRS does not have authorization rules with fine granuality.

3. Protection Poker:

New Requirements:

1. **Patient Hospitalization**
A patient hospitalization module should be included to the OpenMRS system that enables patients to be admitted to a ward with an associated unique tracking number. This module must include the ability to search for available wards and admit patients in them.

2. **Lab Visits**
OpenMRS system must allow entry of each patient’s lab visit information for tests such as X-Rays, blood tests and scans as well as their results.

3. **Patient Log-In**
OpenMRS system must allow patients to create accounts and log onto them so that they can view their medical records.

4. **Notifications to Patients**
OpenMRS system should send email notifications to a patient when their medical records are being accessed. The email should contain who has accessed the medical records and what medical records were accessed along with the time stamp.

5. **Pharmacy Module**
A pharmacy module must be included to the OpenMRS system that enables prescriptions to be automatically filled for a patient after each visit.

Assigning value points to data tables being used in above requirements:

Table	**Value**
visit	5
visit_type	5
visit_attribute_type	40
patient	100
patient_hospitalization**(new)**	40
patient_identifier	2
patient_identifier_type	2
location	1
location_tag_map	1
location_tag	1
location_attribute	2
location_attribute_type	1
encounter	2
encounter_type	2
encounter_provider	2
encounter_role	1
provider	1
provider_attribute	1
provider_attribute_type	1
orders	5
order_type	5
drug_order	5
pharmacy**(new)**	50

Data tables used by requirements:

Requirement	**Data Table Used**	**Table Value points**	**Maximum Value**
Patient Hospitalization	patient_hospitalization	40	100
Lab Visits	Visit, visit_type, visit_attribute_type	50	100
Patient Log-In	patient	100	100
Notifications for Patient Information	Patient_identifier, patient_identifier_type, location, location_tag_map, location_tag, location_attribute, location_attribute_type, encounter, encounter_type, encounter_provider, encounter_role, provider, provider_attribute, provider_attribute_type	20	100
Pharmacy Module	Orders, order_type, drug_order, Visit_type, pharmacy	70	100

Security Risk Calculation:

Team member: Arun

Requirement	**Ease of Attack**	**Value of Assets**	**Security Risk**	**Rank of Security Risk**
Patient Hospitalization	40	40	1600	2
Lab Visits	8	50	400	4
Patient Log-In	20	100	2000	1
Notifications	2	20	40	5
Pharmacy Module	8	70	560	3

Team member: Bhaskar

Requirement	**Ease of Attack**	**Value of Assets**	**Security Risk**	**Rank of Security Risk**
Patient Hospitalization	20	40	800	3
Lab Visits	13	50	650	4
Patient Log-In	40	100	4000	1
Notifications	5	20	100	5
Pharmacy Module	40	70	2800	2

Team member: Ananthram

Requirement	**Ease of Attack**	**Value of Assets**	**Security Risk**	**Rank of Security Risk**
Patient Hospitalization	20	40	800	3
Lab Visits	8	50	400	4
Patient Log-In	40	100	4000	1
Notifications	8	20	160	5
Pharmacy Module	13	70	910	2

Team member: Nivedita

Requirement	**Ease of Attack**	**Value of Assets**	**Security Risk**	**Rank of Security Risk**
Patient Hospitalization	20	40	800	3
Lab Visits	13	50	650	4
Patient Log-In	40	100	4000	1
Notifications	2	20	40	5
Pharmacy Module	20	70	1400	2

Final Table:

Requirement	Ease of Attack	Value of Assets	Security Risk	Rank of Security Risk
Patient Hospitalization	20	40	800	3
Lab Visits	8	50	400	4
Patient Log-In	40	100	4000	1
Notifications	2	20	40	5
Pharmacy Module	20	70	1400	2

Explanation:

We started protection poker by assigning value points to database tables and using them in the requirements. Two new tables: patient_hospitalization and pharmacy need to be created to support the suggested functionalities for OpenMRS system.

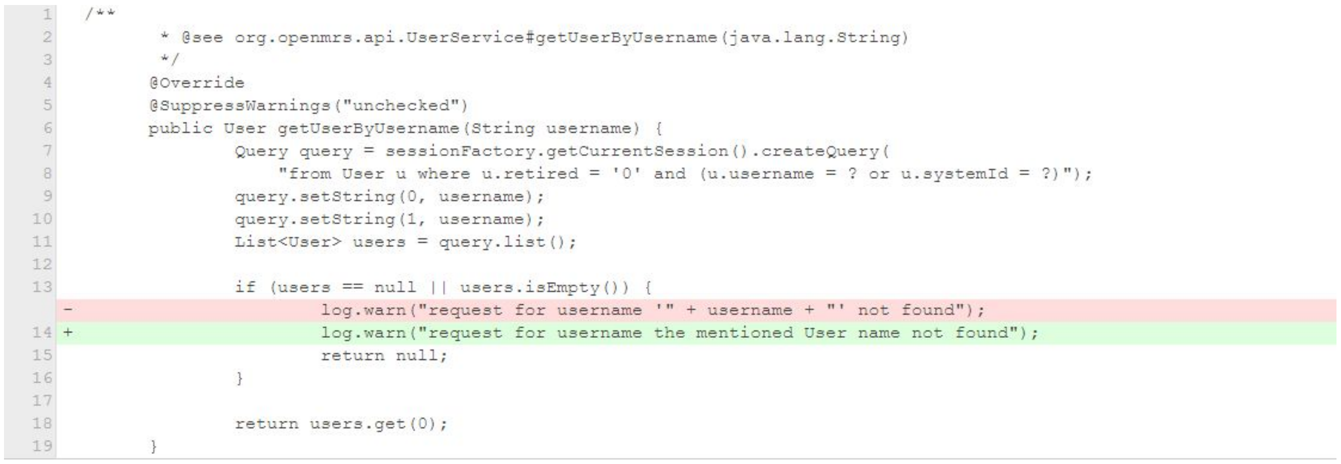
Thereafter each teammate individually assigned ‘‘Ease of Attack Points’ and then calculated Security Risks based on those assigned points and ranked them. Then, we discussed the ‘Ease of Attack’ points from all 4 of our tables to agree on a final ‘Ease of Attack’ points which we then assigned to each new functional requirement. Hence, the game informed all our final rankings depending on each requirement ’s security risk. For example, one teammate had anticipated requirement 1 (Patient Hospitalization) to be much more riskier than requirement 5 (Pharmacy module), while others had a different conclusion. After the game was played, it was obvious that requirement 5 enabled more possible attack vectors into the system, increasing the ease of attack and thus had a significantly higher security risk. Though, requirement 5 was ranked as 2 and requirement 1 was ranked as 3.

Requirement 3 (patient log-in) was found to bear the highest security risk in the system (if implemented) as the ‘Ease of Attack’ points and the value of the patient table in the database was the highest overall. Requirement 2 (lab visits) had access to a valuable part of the database but the ease of attack is quite less. Hence, it poses a lower security risk ranking of 4. Requirement 4 (notifications) was ranked as the least risky of all, as we all agreed that an attack on it will be mostly intangible.

4. Bug Fixes:

1. **Fix #1: To Reduce Log Forging**

File Name:
openmrs-core/api/src/main/java/org/openmrs/api/db/hibernate/HibernateUserDAO.java
File Path:
https://github.com/openmrs/openmrs-core/blob/571f3080b3d870dd7c1d3a00a01aab1fd3fe2295/api/src/main/java/org/openmrs/api/db/hibernate/HibernateUserDAO.java



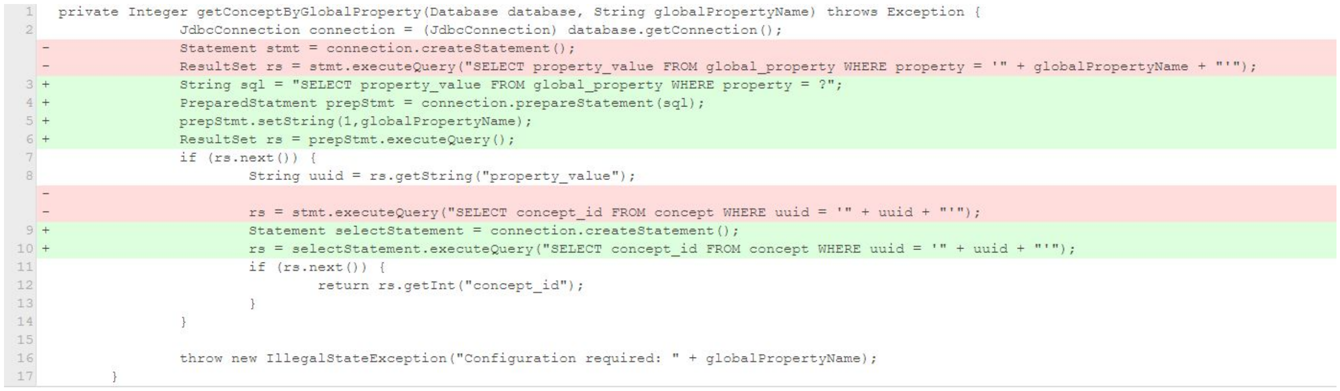
```
  /**
         * @see org.openmrs.api.UserService#getUserByUsername(java.lang.String)
         */
        @Override
        @SuppressWarnings("unchecked")
        public User getUserByUsername(String username) {
                Query query = sessionFactory.getCurrentSession().createQuery(
                    "from User u where u.retired = '0' and (u.username = ? or u.systemId = ?)");
                query.setString(0, username);
                query.setString(1, username);
                List<User> users = query.list();

                if (users == null || users.isEmpty()) {
-                       log.warn("request for username '" + username + "' not found");
+                       log.warn("request for username the mentioned User name not found");
                        return null;
                }

                return users.get(0);
        }
```

Reference: This issue was discovered in the **Issue No. 6** of ***'Issues Identified in OpenMRS using Fortify'*** section of Project Phase 3.

Explanation: Existing lines of code in both the above files introduces the log forging vulnerability. Input that is being received from the user (through form fields/session parameters) is directly written into the log. This also gives the user to feed the form with invalid input, that will be recorded in the log section. The above change makes sure that even if the input is forged, only the necessary information is stored in the Log. This avoids the possibility of crashing the logs with invalid inputs.

<u>Fix #2 : SQL Injection:</u>

File Name:
openmrs-core/api/src/main/java/org/openmrs/util/databasechange/MigrateAllergiesChangeSet.java
File Path:
https://github.com/openmrs/openmrs-core/blob/df4621c1928148d6a3c417c9fdee4004904fb63e/api/src/main/java/org/openmrs/util/databasechange/MigrateAllergiesChangeSet.java



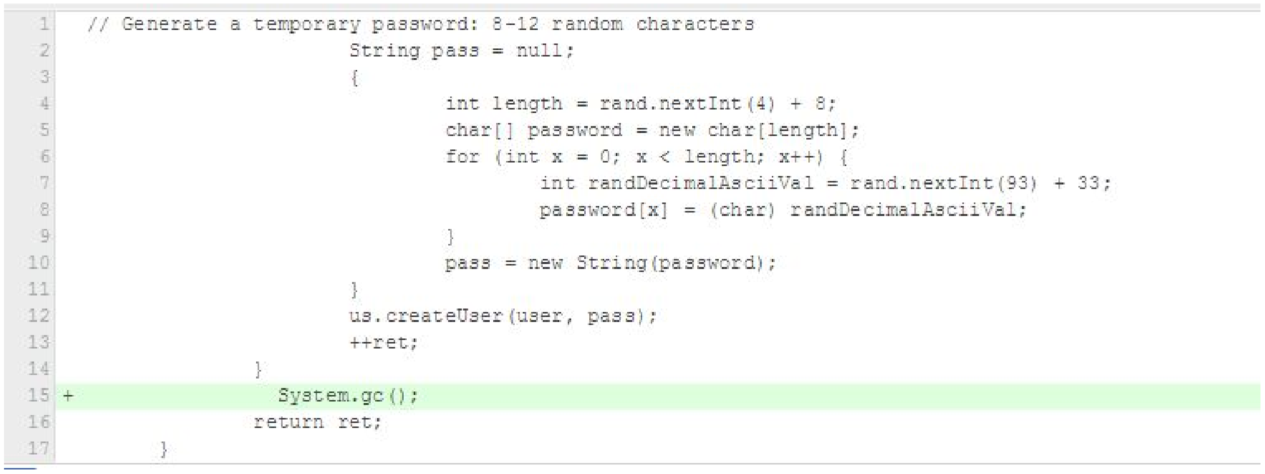
```
  private Integer getConceptByGlobalProperty(Database database, String globalPropertyName) throws Exception {
              JdbcConnection connection = (JdbcConnection) database.getConnection();
-             Statement stmt = connection.createStatement();
-             ResultSet rs = stmt.executeQuery("SELECT property_value FROM global_property WHERE property = '" + globalPropertyName + "'");
+             String sql = "SELECT property_value FROM global_property WHERE property = ?";
+             PreparedStatment prepStmt = connection.prepareStatement(sql);
+             prepStmt.setString(1,globalPropertyName);
+             ResultSet rs = prepStmt.executeQuery();
              if (rs.next()) {
                     String uuid = rs.getString("property_value");
-
-                    rs = stmt.executeQuery("SELECT concept_id FROM concept WHERE uuid = '" + uuid + "'");
+                    Statement selectStatement = connection.createStatement();
+                    rs = selectStatement.executeQuery("SELECT concept_id FROM concept WHERE uuid = '" + uuid + "'");
                     if (rs.next()) {
                            return rs.getInt("concept_id");
                     }
              }

              throw new IllegalStateException("Configuration required: " + globalPropertyName);
       }
```

<u>Reference:</u> This issue was discovered in the **Issue No. 3** of ***'Issues Identified in OpenMRS using Fortify'*** section of Project Phase 3.

<u>Explanation:</u> Existing lines of code in both the above files introduces SQL Injection (OWASP A1) in the application. Prepared Statement is advisable to be used, so that the variables are not exposed and this will ensure that the application remains secure. The changes shown in green, will make sure that SQL injection is prevented in future.

<u>Fix #3 : To mitigate Privacy Violation through Heap Inspection:</u>

File Name:
openmrs-core/api/src/main/java/org/openmrs/migration/MigrationHelper.java
File Path:
https://github.com/openmrs/openmrs-core/blob/73558b0b492b5d1f255844d23658f5d5acb99471/api/src/main/java/org/openmrs/migration/MigrationHelper.java



```
// Generate a temporary password: 8-12 random characters
                        String pass = null;
                        {
                                int length = rand.nextInt(4) + 8;
                                char[] password = new char[length];
                                for (int x = 0; x < length; x++) {
                                        int randDecimalAsciiVal = rand.nextInt(93) + 33;
                                        password[x] = (char) randDecimalAsciiVal;
                                }
                                pass = new String(password);
                        }
                        us.createUser(user, pass);
                        ++ret;
                }
+                 System.gc();
                return ret;
        }
```

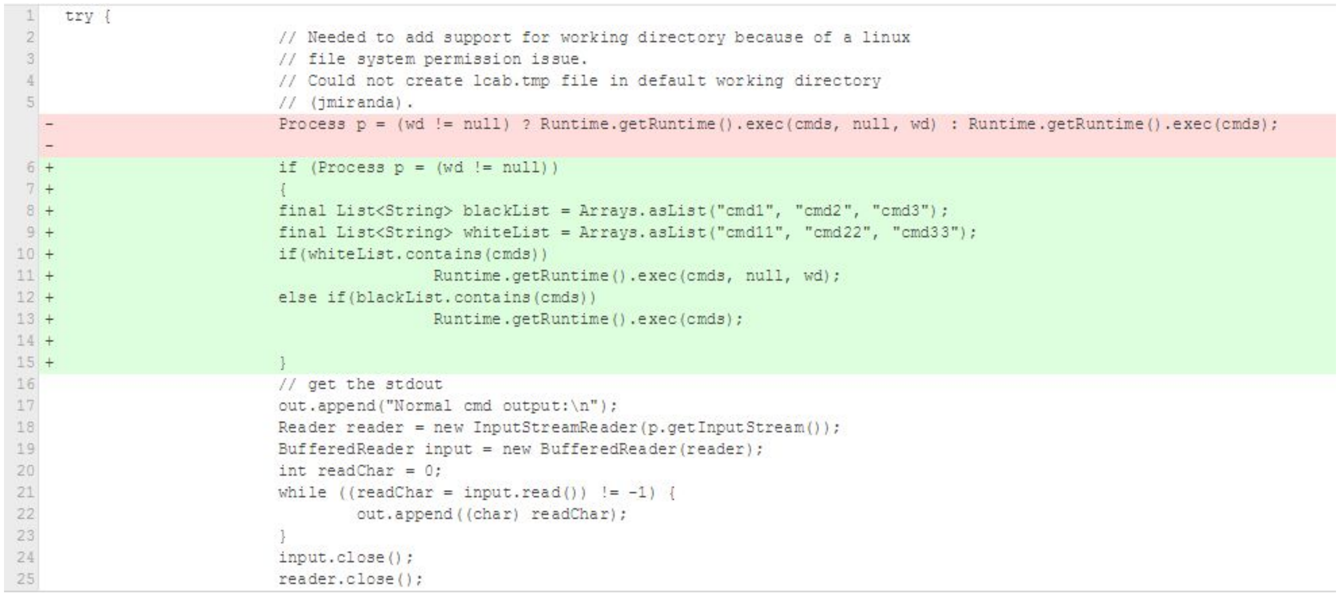
<u>Reference:</u> This issue was discovered in the **Issue No. 8** of ***'Issues Identified in OpenMRS using Fortify'*** section of Project Phase 3.

<u>Explanation:</u>
If an attacker performs heap inspection, the password that is stored may introduce Password Protection Vulnerability. By adding System.gc() function, we ensure that garbage collection takes place explicitly and the password string object is removed from the heap when it goes out of scope. This function is not called within the for loop as it might affect the performance of the application.

<u>Fix #4 : Blacklisting and Whitelisting to Prevent Command Injection:</u>

File Name:
openmrs-core/api/src/test/java/org/openmrs/test/MigrateDataSet.java

File Path:
https://github.com/openmrs/openmrs-core/blob/df148feaddd762ffe7394eabfbdc491eed5e28d0/api/src/test/java/org/openmrs/test/MigrateDataSet.java



```diff
  try {
                              // Needed to add support for working directory because of a linux
                              // file system permission issue.
                              // Could not create lcab.tmp file in default working directory
                              // (jmiranda).
-                             Process p = (wd != null) ? Runtime.getRuntime().exec(cmds, null, wd) : Runtime.getRuntime().exec(cmds);
-
+                             if (Process p = (wd != null))
+                             {
+                             final List<String> blackList = Arrays.asList("cmd1", "cmd2", "cmd3");
+                             final List<String> whiteList = Arrays.asList("cmd11", "cmd22", "cmd33");
+                             if(whiteList.contains(cmds))
+                                             Runtime.getRuntime().exec(cmds, null, wd);
+                             else if(blackList.contains(cmds))
+                                             Runtime.getRuntime().exec(cmds);
+
+                             }
                              // get the stdout
                              out.append("Normal cmd output:\n");
                              Reader reader = new InputStreamReader(p.getInputStream());
                              BufferedReader input = new BufferedReader(reader);
                              int readChar = 0;
                              while ((readChar = input.read()) != -1) {
                                      out.append((char) readChar);
                              }
                              input.close();
                              reader.close();
```

Reference: This issue was discovered in the **Issue No. 2** of ***'Issues Identified in OpenMRS using Fortify'*** section of Project Phase 3.

Explanation:
exec() function is called with a command from untrusted user data which can lead to command injection. To avoid this vulnerability a predetermined set of commands for blacklist and whitelist and depending on which list the command falls in, accordingly execute the command or disallow it. This way, we can ensure that command injection is avoid by allowing only safe commands to execute and inhibiting malicious commands.

Fix #5 : Stored XSS in form:

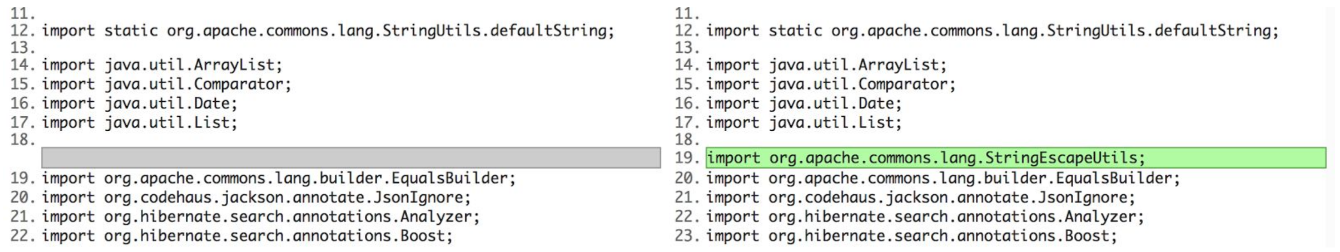
Reference:
This bug is taken from the Test case 1 for Fuzzing With Zap for XSS from project 2.

Explanation:
When patient fills form on patient page, the input is not validated properly. So if javascript code is filled in the form and sent to server it gets stored in the database. Thus, if the patient page is reloaded the script gets executed. This attack can be replicated on many forms in the OpenMRS system.

It could be fixed by using the StringEscapeUtils class in Java. The savePatient function in PatientServiceImpl class is called when patient details are modified. The fix escapes the input in this function before it is written to the database.

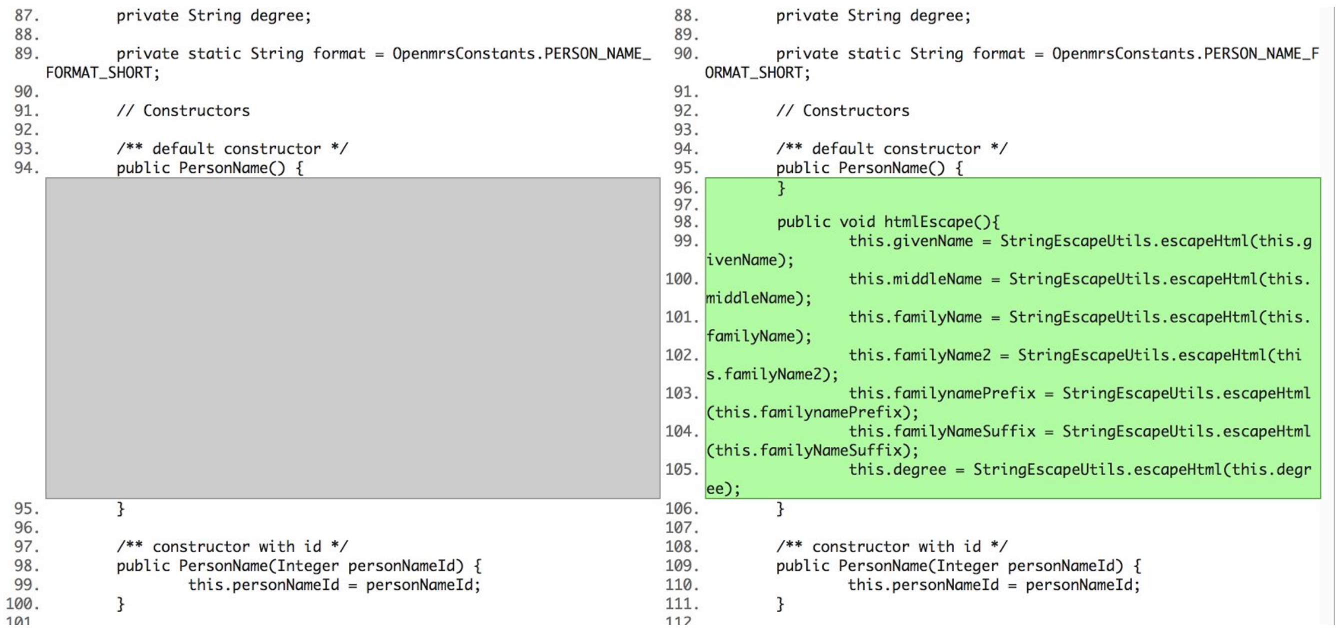
Classes modified:

PersonName.java



```java
import static org.apache.commons.lang.StringUtils.defaultString;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.Date;
import java.util.List;


import org.apache.commons.lang.builder.EqualsBuilder;
import org.codehaus.jackson.annotate.JsonIgnore;
import org.hibernate.search.annotations.Analyzer;
import org.hibernate.search.annotations.Boost;
```

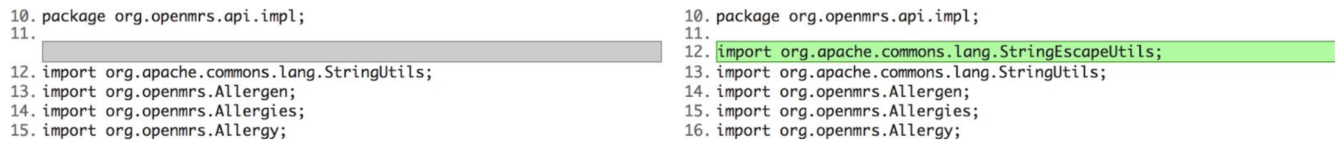
```java
import static org.apache.commons.lang.StringUtils.defaultString;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.Date;
import java.util.List;

import org.apache.commons.lang.StringEscapeUtils;
import org.apache.commons.lang.builder.EqualsBuilder;
import org.codehaus.jackson.annotate.JsonIgnore;
import org.hibernate.search.annotations.Analyzer;
import org.hibernate.search.annotations.Boost;
```



```java
	private String degree;

	private static String format = OpenmrsConstants.PERSON_NAME_FORMAT_SHORT;

	// Constructors

	/** default constructor */
	public PersonName() {
	}

	/** constructor with id */
	public PersonName(Integer personNameId) {
		this.personNameId = personNameId;
	}
```

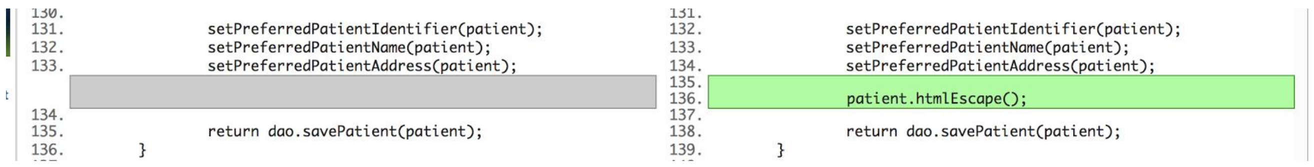
```java
	private String degree;

	private static String format = OpenmrsConstants.PERSON_NAME_FORMAT_SHORT;

	// Constructors

	/** default constructor */
	public PersonName() {
	}

	public void htmlEscape(){
		this.givenName = StringEscapeUtils.escapeHtml(this.givenName);
		this.middleName = StringEscapeUtils.escapeHtml(this.middleName);
		this.familyName = StringEscapeUtils.escapeHtml(this.familyName);
		this.familyName2 = StringEscapeUtils.escapeHtml(this.familyName2);
		this.familynamePrefix = StringEscapeUtils.escapeHtml(this.familynamePrefix);
		this.familyNameSuffix = StringEscapeUtils.escapeHtml(this.familyNameSuffix);
		this.degree = StringEscapeUtils.escapeHtml(this.degree);
	}

	/** constructor with id */
	public PersonName(Integer personNameId) {
		this.personNameId = personNameId;
	}
```

PatientServiceImpl.java.



```java
package org.openmrs.api.impl;


import org.apache.commons.lang.StringUtils;
import org.openmrs.Allergen;
import org.openmrs.Allergies;
import org.openmrs.Allergy;
```

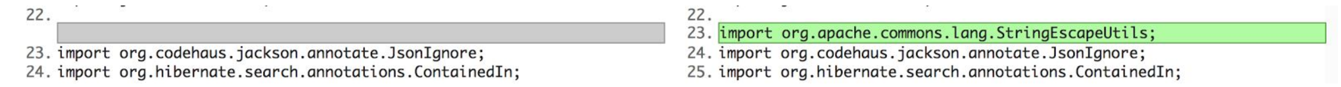
```java
package org.openmrs.api.impl;

import org.apache.commons.lang.StringEscapeUtils;
import org.apache.commons.lang.StringUtils;
import org.openmrs.Allergen;
import org.openmrs.Allergies;
import org.openmrs.Allergy;
```



```java
		setPreferredPatientIdentifier(patient);
		setPreferredPatientName(patient);
		setPreferredPatientAddress(patient);


		return dao.savePatient(patient);
	}
```

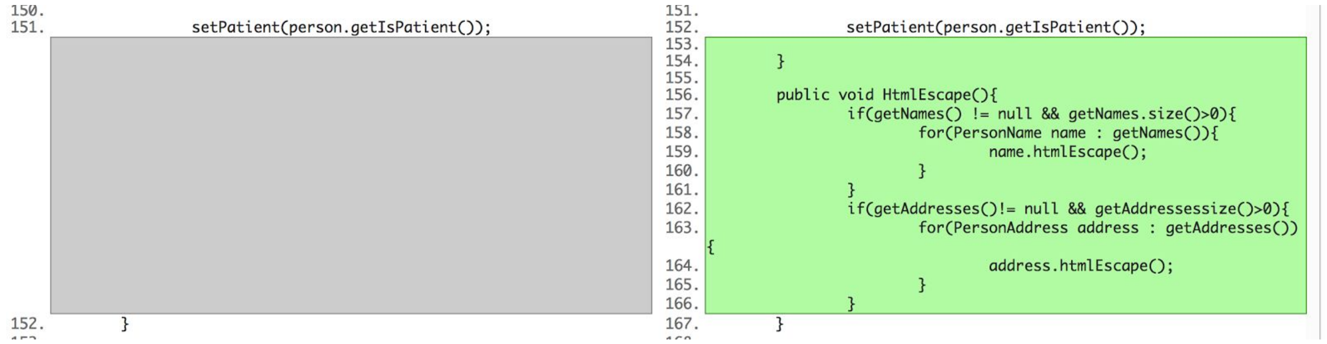
```java
		setPreferredPatientIdentifier(patient);
		setPreferredPatientName(patient);
		setPreferredPatientAddress(patient);

		patient.htmlEscape();

		return dao.savePatient(patient);
	}
```

Person.java



```
import org.codehaus.jackson.annotate.JsonIgnore;
import org.hibernate.search.annotations.ContainedIn;
```

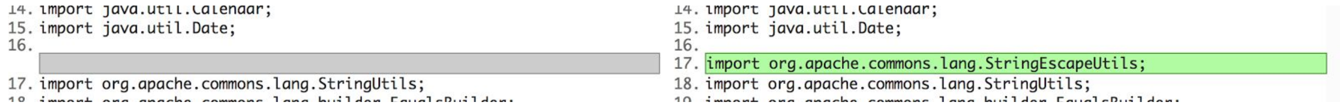
```
import org.apache.commons.lang.StringEscapeUtils;
import org.codehaus.jackson.annotate.JsonIgnore;
import org.hibernate.search.annotations.ContainedIn;
```



```

		setPatient(person.getIsPatient());
	}
```

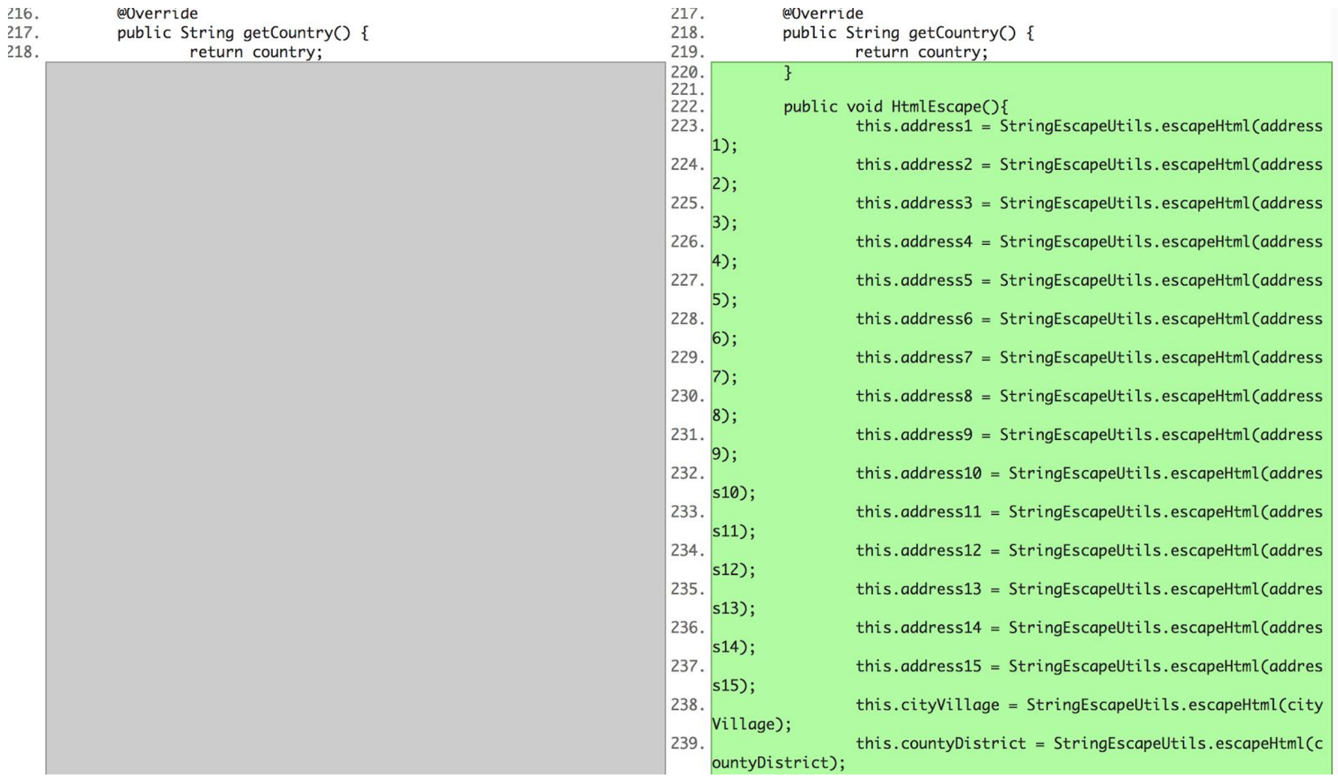
```

		setPatient(person.getIsPatient());

	}

	public void HtmlEscape(){
		if(getNames() != null && getNames.size()>0){
			for(PersonName name : getNames()){
				name.htmlEscape();
			}
		}
		if(getAddresses()!= null && getAddressessize()>0){
			for(PersonAddress address : getAddresses()){
				address.htmlEscape();
			}
		}
	}
```

PersonAddress.java



```
import java.util.Calendar;
import java.util.Date;

import org.apache.commons.lang.StringUtils;
```

```
import java.util.Calendar;
import java.util.Date;

import org.apache.commons.lang.StringEscapeUtils;
import org.apache.commons.lang.StringUtils;
```



```
	@Override
	public String getCountry() {
		return country;
```

```
	@Override
	public String getCountry() {
		return country;
	}

	public void HtmlEscape(){
		this.address1 = StringEscapeUtils.escapeHtml(address1);
		this.address2 = StringEscapeUtils.escapeHtml(address2);
		this.address3 = StringEscapeUtils.escapeHtml(address3);
		this.address4 = StringEscapeUtils.escapeHtml(address4);
		this.address5 = StringEscapeUtils.escapeHtml(address5);
		this.address6 = StringEscapeUtils.escapeHtml(address6);
		this.address7 = StringEscapeUtils.escapeHtml(address7);
		this.address8 = StringEscapeUtils.escapeHtml(address8);
		this.address9 = StringEscapeUtils.escapeHtml(address9);
		this.address10 = StringEscapeUtils.escapeHtml(address10);
		this.address11 = StringEscapeUtils.escapeHtml(address11);
		this.address12 = StringEscapeUtils.escapeHtml(address12);
		this.address13 = StringEscapeUtils.escapeHtml(address13);
		this.address14 = StringEscapeUtils.escapeHtml(address14);
		this.address15 = StringEscapeUtils.escapeHtml(address15);
		this.cityVillage = StringEscapeUtils.escapeHtml(cityVillage);
		this.countyDistrict = StringEscapeUtils.escapeHtml(countyDistrict);
```

```
        }

        /**
         * @param country The country to set.
         */
```

```
ountyDistrict);
                this.stateProvince = StringEscapeUtils.escapeHtml(st
ateProvince);
                this.country = StringEscapeUtils.escapeHtml(countr
y);
                this.postalCode = StringEscapeUtils.escapeHtml(posta
lCode);
        }

        /**
         * @param country The country to set.
         */
```