Preliminary Final Report

Name	Unity ID
Abhash Jain	ajain28
Arjun Sharma	asharm33
Sachin Kumar	skumar26
Aishwarya Sundararajan	asundar2

1) Compilation of report

Executive Summary

The first part of the project deals with identifying test cases that violate the OWASP top 10 application risks. OpenMRS is robust against SQL Injection but has every other security risk that has been captured by OWASP top 10. Broken Authentication, sensitive data exposure, XML entities attack, Broken access control, security misconfiguration, cross-site scripting, insecure deserialization, using components with known vulnerabilities, insufficient logging and monitoring are prevalent in OpenMRS. The test cases for the same have been documented along with necessary mitigation steps.

The second part of the project deals with identifying abuse/misuse cases. OpenMRS lacks in enforcing complex passwords and the code doesn't check for password age and doesn't have a password reuse policy. The abuse/misuse case for find/create patient operation has been documented. The potential threats identified are as follows- Stealing patient information which has a negative impact on the reputation of the hospital , Database delete/flood, Tamper patient vector which falls under the category of both abuse and misuse cases. The attack tree for stealing patient record has been delineated, since a leak in these records could violate GDPR regulations as well as Health Insurance Portability and Accountability Act. Once leaked, these records could reach competitive hospitals or pharma companies and could be exploited in the black market.Brute-force attack, phishing attack, keylogger and penetration test tools are the most common forms of attack on such records.

The third part of the project deals with a running a Fortify check on the OpenMRS code. Several files were scanned and some burning issues were identified: Log forging which could be avoided by making sure that server-controlled values are used instead of client supplied code when logging; Broken access control which allows an unauthorized user to access records can be avoided by handling access control not only in the presentation layer but also in the application and database layers. Path manipulation, privacy violation, password misconfiguration, null dereference, denial of service, and hardcoded password are some of the issues identified with OpenMRS. Adequacy of logging for various operations such as add/edit allergy have been documented. Fuzzing with ZAP proxy was done but no vulnerabilities were found as a result of this testing. Client size bypassing with ZAP failed for two test cases - intercepted or modified password could be updated in the database and intercepted search query results could be updated.The security requirements for OpenMRS have been documented.

The fourth part of the project highlights cases that violate architectural design principles such as complete mediation, fail-safe defaults as well as usable security principles such as appropriate boundaries, explicit authorization etc. Additional functional requirements were found and protection poker was used to identify the most vulnerable feature amongst the added features.

Bug fixes have been implemented for explicit null dereference, unguarded write, resource leak, privacy violation, dereference after null check.

Project Part 1:

Solution 1:

A1: Injection

Test Case ID	Test_A1_1
Test Name	**SQL injection in Login Page**
Steps for test case	i) Go to login page: http://152.46.17.224:8082/openmrs-standalone/login.htm ii) Now provide following credentials on the page username: **admin**, password: **' or ' 1=1** iii) Now after trying the above provided credentials, authentication failed showing message “Invalid username/password. Please try again.”
Expected Result	Login should fail and response should be returned as Invalid username/password. Please try again.
Test Case Status	Passed.
Mitigation by OpenMRS team	Parameterized queries are used along with usage of ORM(Object Relationship Mapping) frameworks like Hibernate.

Test Case ID	Test_A1_2
Test Name	**SQL injection in Find Patient Record**
Steps for test case	i) Login using valid credentials. http://152.46.17.224:8082/openmrs-standalone/login.htm

	ii) Click on Find Patient Record. (http://152.46.17.224:8082/openmrs-standalone/coreapps/findpatient/findPatient.page?app=coreapps.findPatient) iii) In search box enter the query **' union select 1, 1, 1, user(), version() #**, which shows message "**No matching records found**"
Expected Result	It should not return any records through union with another query.
Test Case Status	Passed.
Mitigation by OpenMRS team	Parameterized queries are used along with usage of ORM(Object Relationship Mapping) frameworks like Hibernate.

A2: Broken Authentication

Test Case ID	**Test_A2_1**
Test Name	**Weak password acceptance in Register Patient**
Steps for test case	i) Go to http://152.46.17.224:8082/openmrs-standalone/index.htm, there i was logged in as admin ,then click on admin dropdown top bar of that page,then Myaccount button will appear,so click on it. ii) Next click on Change password button. iii) Now in the page that appears, provide the old password, and provide a weak new password such as **'Test1234'**, and then save it. iv) Now logout and login again with new password
Expected Result	The application should not have allowed to set a weak password while changing the password, as it makes system vulnerable to dictionary and brute-force attacks.
Test Case Status	Failed
Mitigation to be taken by OpenMRS team	The password rules for authentication should be strong and not easily open to dictionary and brute-force attacks. A combination of upper and lower case characters with a mix of alpha-numeric characters that don't usually form a sequence is a good password rule.

Test Case ID	Test_A2_2
Test Name	**Session ids in the url and its generation and rotation after successful login**
Steps for test case	i) Go to http://152.46.17.224:8082/openmrs-standalone/referenceapplication/login.page, fill in the username and password, and also click on inspect in chrome to open network tab's console. ii) Now press login button after filling in login details, and observe in url, session id was not present. iii) Also see the post payload in the network tab of console, there click on login.page row and click on Headers tab in right side window, there observe the session id generated and no-caching done as evident by following data: Cache-Control: max-age=0 Connection: keep-alive Cookie: referenceapplication.lastSessionLocation=4; _REFERENCE_APPLICATION_LAST_USER_=92668751; JSESSIONID=F315C3309D0593996E8279B7FC6AE7CC; referenceapplication.lastSessionLocation=4; _REFERENCE_APPLICATION_LAST_USER_=92668751 iv) Now logout and repeat steps i to iii, to see the cookie details again Cache-Control: max-age=0 Connection: keep-alive Cookie: referenceapplication.lastSessionLocation=4; _REFERENCE_APPLICATION_LAST_USER_=92668751; JSESSIONID=6B42EDD70DE4CC5F85FDBB1B0B85AC39; referenceapplication.lastSessionLocation=4; _REFERENCE_APPLICATION_LAST_USER_=92668751 v) Now as you observe session id's are rotated and change and randomized for every new session.
Expected Result	Session id's should not be transmitted in plain text as part of url and session ids should be rotated and randomized for every session.
Test Case Status	Passed , because session id's are not transmitted in plain text as part of url and session and session id's are randomized for every session
Mitigation by OpenMRS team	Secure built-in session manager used to generate random session ids and also sessions ids are included in the post payload but not in the url

A3: Sensitive Data Exposure

Test Case ID	Test_A3_1
Test Name	**Data encryption in HTTP data transmission**
Steps for test case	i) Go to http://152.46.17.224:8082/openmrs-standalone/registrationapp/registerPatient.page?appId=referenceapplication.registrationapp.registerPatient to register a patient. ii) Fill in the details in all fields and click on Confirm button in confirm tab to submit. iii) In Chrome browser,right click on one of the fields and click inspect and in the open window click on Network tab iii) Then submit the page and notice in the network console many entries now have appeared iv) Now click on the link http://152.46.17.224:8082/openmrs-standalone/coreapps/clinicianfacing/patient.page?patientId=dc00030d-5ff0-4367-bf27-005b4256f6be appearing in the network console, once you click on it ,then click on the Preview tab in the subwindow that appears ,so there you will be able to see all the details submitted by the user being in clear text in the POST request payload. As in that payload all the details were in plaintext : TestGiven pMiddle 2Family Name Male 27 year(s) (02.Feb.1991) Edit Show Contact Info Hide Contact Info Test address, RaleighNCUS27607 Address 919112345 Telephone Number Edit Patient ID 1003EY
Expected Result	Submitted patient details after registration should not have appeared in the post payload as plain text, since a attacker can execute man in middle attack while capturing the payload data at transport layer.
Test Case Status	Failed
Mitigation to be taken by OpenMRS	The data submitted in the post payload must be encrypted to prevent a man in the middle attack.

Test Case ID	Test_A3_2
Test Name	**Checking presence of Browser Security directives**
Steps for test case	i) Go to http://152.46.17.224:8082/openmrs-standalone/login.htm and provide the details as admin and

	Admin123. ii) Now right click on page and click on Inspect and navigate to Network tab in the window that appear. iii) Now hit the log in button. iv) Now in the Network tab click on home.page in the pages that appear in Name column and then click on headers tab in the corresponding subwindow that appears on left. v) Now in the headers data that appears in the subwindow look for the ContentSecuritypolicy or CSP which is usually declared by <meta http-equiv="Content-Security-Policy" content="default-src 'self'; img-src https://*; child-src 'none';"> vi) As we observed we didn't found in the headers or page source any security policy directive being used in the openmrs which makes it prone to XSS and injection attacks.
Expected Result	There should have been a ContentSecurityPolicy or any other browser security policy directive should have been implemented in the OpenMRS.
Test Case Status	Failed

A5 : Broken Access Control

Test Case ID	**Test_A5_1**
Test Name	**Acting as admin when logged in as low-privilege user**
Steps for test case	i)Go to Admin Home Page using Admin credential. (URL: http://152.46.17.224:8082/openmrs-standalone/referenceapplication/home.page) ii) Click on System Administration on the home page and then click on"Advanced Administration". iii) On the next page Select "Manage Users" Anchor link. iv) From the Role drop-down menu select "Organizational: Nurse" as roles. It will list Nurse user Select the hyperlink under System-ID **4-2**.that brings to next page. v) Change the username and password from the input item User password's and confirm it in the next password box the same Password. I have selected the password "Ncsu1234".

	And select the save user button to save the information. vi) Now login as nurse, then click on Appointment scheduling,where you don't have access to manage service type, as this privilege was available to admin but not available for this role iii)But you open a new tab and openManage Service types page http://152.46.17.224:8082/openmrs-standalone/appointmentschedulingui/manageAppointmentTypes.page , then it will open where you can click on New service type button.
Expected Result	Nurse role didn't had access to create new service type as that page was not available for that role in UI ,so manage services pages shouldn't have appeared for nurse role
Test Case Status	Failed
Mitigation to be taken by OpenMRS	Various level of users of the application should be authorized to access different parts of the application after authentication.

Test Case ID	**Test_A5_2**
Test Name	**Elevation of privilege of standard user**
Steps for test case	i)Go to Admin Home Page using Admin credential. (URL: http://152.46.17.224:8082/openmrs-standalone/referenceapplication/home.page) ii) Click on System Administration on the home page and then click on"Advanced Administration". iii) On the next page Select "Manage Users" Anchor link. iv) From the Role drop-down menu select "Organizational: Nurse" as roles. It will list Nurse user Select the hyperlink under System-ID 4-2.that brings to next page. v) Change the username and password from the input item User password's and confirm it in the next password box the same Password. I have selected the password "Ncsu1234". And select the save user button to save the information. vi) Now login as nurse,where on the dashboard you don't have access to System administration option. vii) Now open manage users link which is available to admin user, http://152.46.17.224:8082/openmrs-

	standalone/admin/users/users.list?name=admin&role=&action=Search. viii) Now type admin in find user or name field, in search result admin records will appear, click on admin link appearing in first column of the record of the search result. viiii) Now in the page that appear, we can change the admin;s password and other details,so let's change the password from Admin123 to Admin1234 and save it ix) Now logout and login with username as admin and password as Admin123, it will fail and now login with password Admin1234. x) So here , a standard user with not having access to manage the users did changed the admin account details.
Expected Result	User 'nurse' not having access to admin pages of manage users, shouldn't have been able to access it and ability to also modify the data over there
Test Case Status	Failed
Mitigation to be taken by OpenMRS	Various level of users of the application should be authorized to access different parts of the application after authentication.

A6 : Security Misconfiguration

Test Case ID	Test_A6_1
Test Name	**Unauthorized access to Admin Role**
Steps for test case	i) Go to Admin Home Page using Admin credential. (URL: http://152.46.17.224:8082/openmrs-standalone/referenceapplication/home.page) ii) Click on System Administration on the home page and then click on"Advanced Administration". iii) On the next page Select "Manage Users" Anchor link. iv) From the Role drop-down menu select "Organizational: Nurse" as roles. It will list Nurse user Select the hyperlink under System-ID **4-2**.that brings to next page. V) Change the username and password from the input item User password's and confirm it in the next password box the same Password. I have selected the password "Ncsu1234". And select the save user button to save the information. Vii) Logout from admin role and login as username "nurse" and password which

	you have given in your password field. My case password was “Ncsu1234”. Viii) Now you have access to the nurse Dashboard which comes with limited access. Ix) Now login in another browser as Admin and copy the “Advanced Administration page” URL link and copy it to the browser where you have login as nurse. (URL : http://152.46.17.224:8082/openmrs-standalone/admin/index.htm) X) Now you see that you are able to access the Admin page, even though you was having limited role access.
Expected Result	When you login as nurse you should not be allowed the page of Admin only. We should get a suitable error message on the page when you try such link.
Test Case Status	Failed
Mitigation to be taken by OpenMRS	When a non-admin accesses an admin URL, an error message reading - “Access is denied” should be displayed. Appropriate checks should be placed to make sure that the level of user accessing the URL is authorized to do so.

Test Case ID	**Test_A6_2**
Test Name	**Unable to handle invalid urls**
Steps for test case	i) Login to portal as admin. ii) Navigate to System Administration > Advanced Administration > View Log Entries. iii) Click on Search then you’ll get the list of log entries. iv) change value of generatedBy parameter to any invalid value for eg. http://152.46.17.224:8082/openmrs-standalone/module/idgen/viewLogEntries.list?source=&identifier=&fromDate=&toDate=&comment=&generatedBy=asdbhjsfs&action=Search v) Instead of getting an error such as “No Log Entries Found”, we get complete error stack trace “HTTP Status 500 Request processing failed; nested exception is TypeMismatchException”.
Expected Result	The application should be able to handle invalid urls and not throw error stack trace.
Test Case Status	Failed

A7: Cross-site scripting

Test Case ID	Test_A7_1
Test Name	**Reflected XSS in EditProfile**
Steps for test case	i) Go to Find patient page http://152.46.17.224:8082/openmrs-standalone/coreapps/findpatient/findPatient.page?app=coreapps.findPatient ii) Now click on the row for patient Paul Walker iii) Now click on Edit link on the page that appears. iv) Now click on Name link appearing on left sidebar, followed by which fill in details “</script><script>alert(1234)</script><script>” in the Middle name section and then on Save form on top of the page and finally click on Confirm button on the page that appears. v) Now again Paul Walker profile details page appear,so click on Edit button on top of the page. vi) Now alert box will popup showing message as ‘12345’, click on OK and then you will reach again back to edit page with all the details.
Expected Result	Script code should not have been accepted as a valid value for the Middle name field in edit profile field,as it allowed the script execution to print some value.
Test Case Status	Failed

Test Case ID	Test_A7_2
Test Name	**Check stored XSS in creating Service types**
Steps for test case	i) Go to Manage Service types page http://152.46.17.224:8082/openmrs-standalone/appointmentschedulingui/manageAppointmentTypes.page and then click on New service type button. ii) Now in service name, type in " onmouseover="('page hacked')" and fill other parameters and click on save to save that record. iii) Now go to http://152.46.17.224:8082/openmrs-standalone/appointmentschedulingui/home.page and then again go to service type pagehttp://152.46.17.224:8082/openmrs-standalone/appointmentschedulingui/manageAppointmentTypes.page

	iv) Now do a mouseover on the page and you won't notice the alert as per the script we injected in the service name.
Expected Result	There shouldn't be an alert message popup for the script injected in the name field of new service type
Test Case Status	Passed
Mitigation by OpenMRS team	Name input field had been sanitized before loading it.

A10: Insufficient logging and monitoring

Test Case ID	**Test_A10_1**
Test Name	**Server Log to list entry and exit point of function**
Steps for test case	i) From Admin Home page navigate to "System Administration" tab. URL for admin page is (http://152.46.17.224:8082/openmrs-standalone/index.htm). ii) In System Administration page if you select "Advanced administration" and select "View Server log". iii) Here you can see that Server log are very detailed and tell us about the Exit and entry point in method.
Expected Result	Whenever a user access any code these information will be logged and can be view by administration. Using this information they can found out that what is most used method. This can help us to find the hot code point .
Test Case Status	Passed

Test Case ID	**Test_A10_2**
Test Name	**Logging the login information and all the Server based transactions and events**
Steps for test case	i) Go to http://152.46.17.224:8082/openmrs-standalone/admin/maintenance/currentUsers.list to see the list of logged in users in application. ii) Also go to http://152.46.17.224:8082/openmrs-

	standalone/admin/maintenance/serverLog.form to view the server log to see all the application based transactions and events logged there which did provide a comprehensive view to monitor the suspicious events or all the modules accessed by respective users
Expected Result	Application should log the logged in users and all the applications based transactions or events for monitoring the suspicious activity.
Test Case Status	Passed
Actual Result	Application should log the logged in users and all the applications based transactions or events for monitoring the suspicious activity. This data can be used for auditing purpose.
Mitigation by OpenMRS team	Audit trail maintained to log high value transactions and also a view to see the logged in users to monitor the suspicious activity

Solution 2:

OWASP dependency check report attached with it.

Solution 3:

A4 : XML External Entities (XXE)

Test Case ID	Test_A4_1
Test Name	**Apache Standard Taglibs (standard-1.1.2.jar)**
Vulnerabilities	Apache Standard Taglibs before 1.2.3 allows remote attackers to execute arbitrary code or conduct external XML entity (XXE) attacks via a crafted XSLT extension in a (1) <x:parse> or (2) <x:transform> JSTL XML tag.
CVE Search List	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-0254
Mitigation Steps	Users should upgrade to latest version of standard taglibs library.

Test Case ID	Test_A4_2
Test Name	**Apache POI (poi-3.12.jar)**
Vulnerabilities	Apache POI is vulnerable to a denial of service, cause by an XML External Entity Injection (XXE) error when processing XML data. By using a specially-crafted OOXML file, a remote attacker could exploit this vulnerability to consume all available CPU resources
CVE Search List	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5644
Mitigation Steps	Users with applications which accept content from external or untrusted sources are advised to upgrade to Apache POI 3.15 or newer.

A8 : Insecure Deserialization

Test Case ID	Test_A8_1
Test Name	**Jackson (jackson-databind-2.8.1.jar)**
Vulnerabilities	A deserialization flaw was discovered in the jackson-databind in versions before 2.8.10 and 2.9.1, which could allow an unauthenticated user to perform code execution by sending the maliciously crafted input to the readValue method of the ObjectMapper. This issue extends the previous flaw CVE-2017-7525 by blacklisting more classes that could be used maliciously.
CVE Search List	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-15095
Mitigation Steps	Mitigation to this problem is to not trigger polymorphic deserialization globally by using: objectMapper.enableDefaultTyping() and rather use @JsonTypeInfo on the class property to explicitly define the type information. For more information on this issue please refer to https://www.github.com/mbechler/marshalsec/blob/master/marshalsec.pdf?raw=true

Test Case ID	Test_A8_2
Test Name	**Spring JMS (spring-jms-3.0.5.RELEASE.jar)**
Vulnerabilities	Spring Framework 3.0.0 through 3.0.5, Spring Security 3.0.0 through 3.0.5 and 2.0.0 through 2.0.6, and possibly other versions deserialize objects from untrusted sources, which allows remote attackers to bypass intended security restrictions and execute untrusted code by (1) serializing a java.lang.Proxy instance and using InvocationHandler, or (2) accessing internal AOP interfaces, as demonstrated using deserialization of a DefaultListableBeanFactory instance to execute arbitrary commands via the java.lang.Runtime class.
CVE Search List	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-2894
Mitigation Steps	Users should migrate away from serialization-based remoting in cases where the client cannot be trusted, as it is a potential source of vulnerabilities in both Spring and non-Spring applications. All users may mitigate this issue by upgrading to Spring Framework 3.0.6 and Spring Security 3.0.6. Spring Framework users should make use of the additional features introduced to prevent deserialization of malicious proxies.

Solution 4:

A9 : Using Components with Known Vulnerabilities

Test Case ID	Test_A9_1
Test Name	**Apache Solr Core (solr-core-4.10.4.jar)**
Vulnerabilities	Cross-site scripting (XSS) vulnerability in webapp/web/js/scripts/plugins.js in the stats page in the Admin UI in Apache Solr before 5.3.1 allows remote attackers to inject arbitrary web script or HTML via the entry parameter to a plugins/cache URI.
CVE Search List	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-8797

Test Case ID	Test_A9_2
Test Name	**Hibernate Bean Validation (hibernate-validator-4.2.0.Final.jar)**
Vulnerabilities	In Hibernate Validator 5.2.x before 5.2.5 final, 5.3.x, and 5.4.x, it was found that when the security manager's reflective permissions, which allows it to access the private members of the class, are granted to Hibernate Validator, a potential privilege escalation can occur. By allowing the calling code to access those private members without the permission an attacker may be able to validate an invalid instance and access the private member value via ConstraintViolation#getInvalidValue().
CVE Search List	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-7536

References :
i) https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2017_Project for top ten vulnerabilities
ii) https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP for understanding browser security policy directives
iii) https://issues.openmrs.org/secure/Dashboard.jspa to see what all issues are open ,so that instead find a new one rather than finding a existing one.
iv) https://cxsecurity.com/issue/WLB-2011100117 for mitigations steps for Spring JMS issue.
v) https://poi.apache.org/ for mitigations steps for Apache POI issue.

Project Part 2:

Solution 1:

OpenMRS password policy is as follows:

Maximum password length :
There is no upper limit specified in the OpenMRS code or properties for setting the passwords. Password is stored as hash of 128 characters in database, however in our tests application allowed saving passwords greater than 128 characters.

Minimum password length :

As per the defaults ,minimum password length is 8 characters and it can be modified by changing the value of system property **“security.passwordMinimumLength“**

Allowed characters :

Uppercase characters,lowercase characters,atleast one digit, atleast one non-digit

Number of allowed character categories:

uppercase letter - atleast 1
lowercase letter - atleast 1
numbers - atleast 1
Symbols - any

Password age :

There are no properties in the OpenMRS and also no checks in the code checking the password age.

Password reuse policy :

There are no properties in the OpenMRS and also no checks in the code checking the password reuse. We have tried to Manually set one of the old password and openMRS has allowed us to setup the same old password.

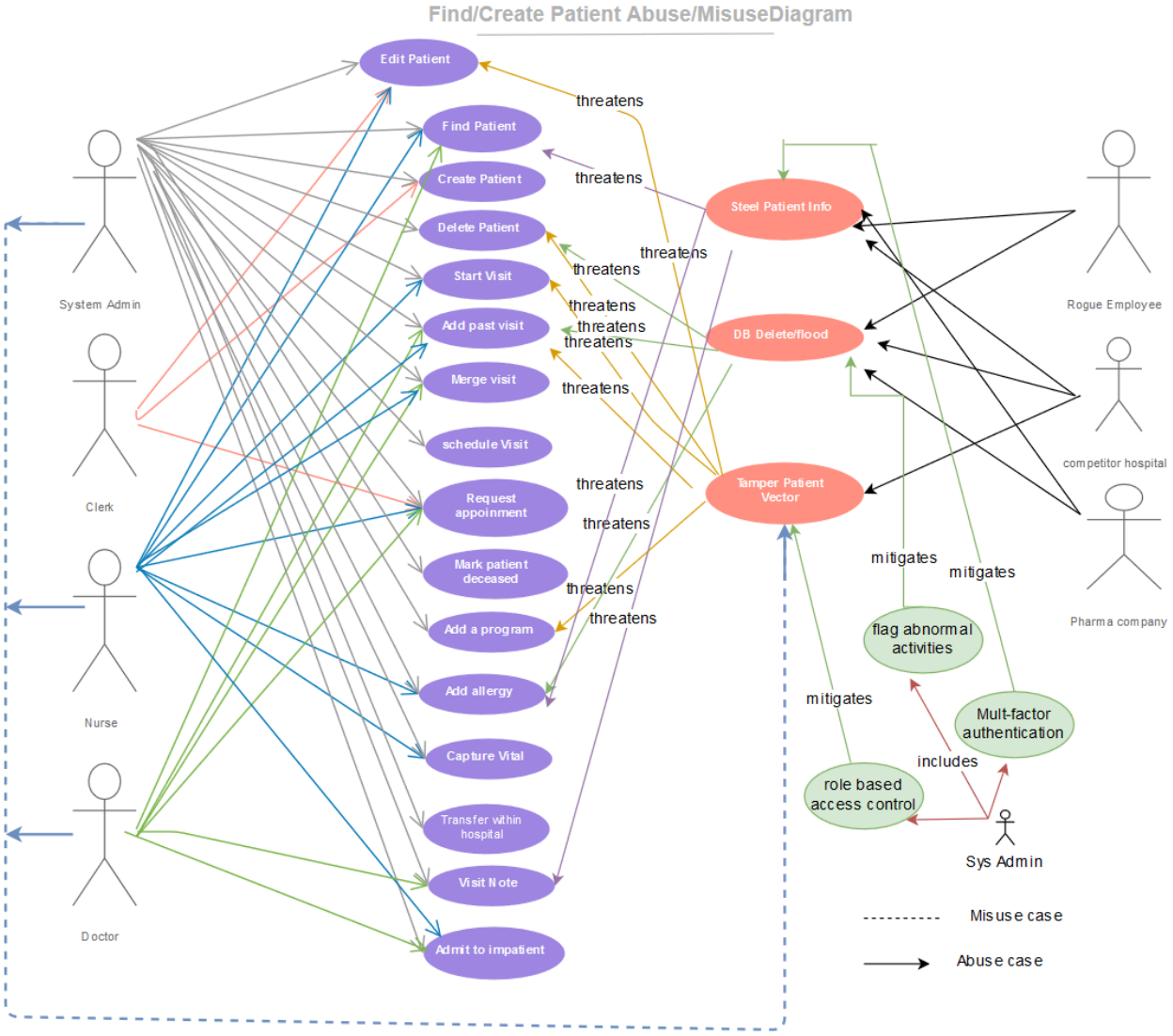
Account lock out:

As per OpenMRS documentation,default number of incorrect login before lockout is 7, however it can be changed by changing the value of property **“security.allowedFailedLoginsBeforeLockout”.** After max number of incorrect logins ,users are locked out for 5 minutes. Ip address are blocked after 10 incorrect username/password attempts.

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

Solution 2:

Abuse/ Misuse Case:



1. Abuse Case : SPI Description:

- **Name :** Steal Patient Information
- **Summary :** Patient details are extremely sensitive and someone can do a lot of harm by stealing the patient details.Possible consequences will be data sold to marketers, who then use it to spam the patients.
- **Author :** Sachin Kumar
- **Date :** 10/1/2018
- **Basic Path:**

BP0 : If a username is known, attacker like competitor company or pharma company can try to guess the password with brute-force attacks. Once password is found,attacker can login into the system and can steal patient details.

BP1 : A rogue employee, which can be system admin or clerk or patient or doctor, can login into the system with their credentials and steal the patient details to use it in malicious way.

- **Alternate Path:**

AP1 : An attacker can obtain password through phishing or social engineering., other than hacking the password

AP2 : An attacker after doing identity theft of a patient, can successfully trick nurse or doctor to reveal the patient details.

- **Capture Points:**

CP1 : Patients are authenticated with multi-factors authentication like sending an One time password to email and mobile phone in addition to patient id check before revealing the patient details on request by a patient. (AP2)

CP2 : Account lockouts after number of failed login attempts, followed by a cooling off period before allowing relogin, prevent attackers usage of brute-force methods to guess password.

- **Extension Points:** None

- **Preconditions:**

PC1 : System has users mapped to roles like admin,doctor,patient,clerk and then those users are then mapped to some application and organisational privileges.

PC2 : System allows login of different roles like admin,doctor,patient,clerk into the OpenMRS system.

- **Assumptions:**

AS1 : User being manipulated by social engineering to reveal the openmrs password.

AS2 : Users like doctor,clerk,nurse and admin has access to certain account privileges with certain permissions to view or edit patient details.

- **Worst Case Threat (Postcondition):**

  WS1 : All of the patient details of OpenMRS system has been compromised

- **Capture Guarantee(Postcondition):**

  CG1 : Patient details data is secure and not hackable by the attacker.

- **Related Business Rules:**

BR1 : Admin,nurse and clerk have access to view the patient data.

- **Potential Misuser Profile:**

Someone with the knowledge of hacking and motivation or incentive to attack and misuse the stolen patient details.

- **Stakeholders and Threats:**

SH1 Doctor/Clerk/Nurse : These roles are under threat as their accounts has been compromised to hack into the openmrs system.

SH2 Patient : Since its patients details that has been hacked, they are the stakeholders with most impact.

SH3 Hospital : Reputation of Hospital among its patients will bear a large negative impact, impairing the trust among the patients who have entrusted their personal details with them.

- **Scope:** Find/Create Patient module

- **Abstraction Level:** Attacker goal

- **Precision Level:** Focused

2. Abuse Case : DBDF Description:

- **Name :**Database Delete/Flood

- **Summary :** Since different users of OpenMRS has access to enter patient details into the system, which is then stored into database, it is highly likely that someone with malicious intent making use of UI interface ,can delete all the records stored in database or flood database with fake patient details

- **Author :** Arjun Sharma

- **Date :** 10/2/2018

- **Basic Path:**

BP0 : For a username belonging to the roles of either of admin,doctor and nurse is known with the access to enter or edit the patient details, then attacker can try to guess the password with brute-force attacks. Once password is found,attacker can delete all the patient details or enter the fake patient details.

- **Alternate Path:**

AP1 : An attacker can obtain password through phishing or social engineering., other than hacking the password

AP2 : An attacker after doing identity theft of a patient, can successfully trick nurse or doctor to reveal the patient details.

- **Capture Points:**

CP1 : Patients are authenticated with multi-factors authentication like sending an One time password to email and mobile phone in addition to patient id check before revealing the patient details on request by a patient. (AP2)

CP2 : Account lockouts after number of failed login attempts, followed by a cooling off period before allowing relogin, prevent attackers usage of brute-force methods to guess password.

CP3 : Automated monitoring of abnormal activities like larger number of changes to patient details over shorter time period, larger number of edits done by a user for the patients can be be used as precursor to automated flagging or lockdown of that user account.

- **Extension Points:** None

- **Preconditions:**

PC1 : System has users mapped to roles like admin,doctor,patient,clerk and then those users are then mapped to some application and organisational privileges.

PC2 : System allows login of different roles like admin,doctor,patient,clerk into the OpenMRS system.

- **Assumptions:**

AS1 : User being manipulated by social engineering to reveal the openmrs password.

AS2 : Users like doctor,clerk,nurse and admin has access to certain account privileges with certain permissions to enter or edit patient details.

- **Worst Case Threat (Postcondition):**

WS1 : All of the patient details of OpenMRS system has been wiped out from the system.

WS2 : Fake records has flooded the database storage and subsequent exhaustion of storage ,resulting in application availability for real users.

- **Capture Guarantee(Postcondition):**

CG1 : Patient details data is secure to be deleted by the attacker.

CG2 : Application is secure to the crash because of the flooding of database possible by fake data insertion by attacker.

- **Related Business Rules:**
  BR1 : Admin and doctor have access to tamper patient data.

- **Potential Misuser Profile:**
Someone with the motivation or incentive to tarnish the reputation of the functioning medical hospital.

- **Stakeholders and Threats:**
SH1 Doctor/Clerk/Nurse : These roles are under threat as their accounts has been compromised to the competitive employees.

SH2 Patient : Since its patients details that has been hacked, they are the stakeholders with most impact.

SH3 Hospital : Reputation of the hospital among its patients will bear a large negative impact, impairing the trust among the patients who have entrusted their personal details with them.

- **Scope:** Find/Create Patient module

- **Abstraction Level:** Attacker goal

- **Precision Level:** Focused

3. Abuse Case : TPV Description:

- **Name :**Tamper patient vector - Abuse

- **Summary :** Competitive employees have a motive to tamper with existing patient records in order to disrepute the competing hospital. If patient records are tampered and if the diagnosis recorded for patients are changed, they could be treated for the wrong disease. The incentive is for the patients to leave the current hospital with tampered records and seek medical attention at competitive hospitals.

- **Author :** Aishwarya Sundararajan

- **Date :** 10/3/2018

- **Basic Path:**

BP0 : For a username belonging to the roles of either of admin,doctor and nurse is known with the access to enter or edit the patient details, then attacker can try to guess the password with brute-force attacks. Once password is found,attacker can tamper with the patient details.

- **Alternate Path:**

AP1 : A competitive employee can threaten Admins or doctors to obtain their credentials

AP2 : After obtaining the credentials, the employee can enter the system and change patient details and diagnosis.

- **Capture Points:**

CP1 : Admin, doctors, and nurses are authenticated with multi-factors authentication like sending a One time password to email and mobile phone in addition to verifying their id before revealing any details. If any one of these authentications fail, fail-safe default is the way to go.

CP2 : Automated monitoring of abnormal activities like larger number of changes to patient details over shorter time period, larger number of edits done by a user for the patients can be be used as precursor to automated flagging or lockdown of that user account.

- **Extension Points:** None

- **Preconditions:**

PC1 : System has users mapped to roles like admin,doctor,patient,clerk and then those users are then mapped to some application and organisational privileges.

PC2 : System allows login of different roles like admin,doctor,patient,clerk into the OpenMRS system.

- **Assumptions:**

AS1 : Admin, Nurse, or Doctor could be bribed or threatened to provide their access credentials.

AS2 : Users like doctor and admin has access to certain account privileges with certain permissions to edit patient details.

- **Worst Case Threat (Postcondition):**

WS1 : All of the patient details of OpenMRS system have been tampered. None of the records are accurate. The entire medical system is down.

WS2 : Loss of trust from patients' perspective. Employees could lose their jobs for good.

- **Capture Guarantee(Postcondition):**

CG1 : Patient details data is secure to be tampered by the competitive employee..

- **Worst Case Threat (Postcondition):**

  WS1 : Many patient records of OpenMRS system have been tampered.

WS2 : Loss of trust from patients' perspective. Employees could lose their jobs for good.

- **Capture Guarantee(Postcondition):**

  CG1 : A confirmation alert box just before changing the patients' records may prevent the admin from changing the wrong detail

  CG2: The admin cannot submit wrong information

- **Related Business Rules:**

  BR1 : Admin,nurse and clerk have access to enter/delete patient data.

- **Potential Misuser Profile:**

Someone with the knowledge of hacking and motivation or incentive to crash the application by flooding the database or deleting the patient details.

- **Stakeholders and Threats:**

SH1 Doctor/Clerk/Nurse : These roles are under threat as their accounts has been compromised to hack into the openmrs system.

SH2 Patient : Since its patients details that has been hacked, they are the stakeholders with most impact.

SH3 Hospital
: Reputation of Hospital among its patients will bear a large negative impact, impairing the trust among the patients who have entrusted their personal details with them.

- **Scope:** Delete patient/ Start visit/Add program/Add past visit modules

- **Abstraction Level:** Competitive employee goal

- **Precision Level:** Focused

4. Misuse Case : TPV-Misuse Description:

- **Name :**Tamper patient vector - Misuse

- **Summary :** System administrator could accidentally tamper patient records or add wrong programs which could affect patient details and diagnosis. This could affect the drugs being administered to the patient and in the worst case cause the patient's condition to worsen.

- **Author :** Abhash Jain

- **Date :** 10/3/2018

- **Basic Path:**

BP0 : For a username belonging to the roles of either of admin,doctor and nurse is known with the access to enter or edit the patient details, then attacker can try to guess the password with brute-force attacks. Once password is found,attacker can tamper with the patient details.

- **Alternate Path:**

AP1 : An attacker can obtain password through phishing or social engineering., other than hacking the password

AP2 : An attacker after doing identity theft of a patient, can successfully trick nurse or doctor to reveal the patient details.

- **Capture Points:**

CP1 : Patients are authenticated with multi-factors authentication like sending an One time password to email and mobile phone in addition to patient id check before revealing the patient details on request by a patient. (AP2)

CP2 : Account lockouts after number of failed login attempts, followed by a cooling off period before allowing relogin, prevent attackers usage of brute-force methods to guess password.

CP3 : Automated monitoring of abnormal activities like larger number of changes to patient details over shorter time period, larger number of edits done by a user for the patients can be be used as precursor to automated flagging or lockdown of that user account.

- **Extension Points:** None

- **Worst Case Threat (Postcondition):**

WS1 : All of the patient details of OpenMRS system have been tampered. None of the records are accurate. The entire medical system is down.

WS2 : Loss of trust from patients' perspective. Employees could lose their jobs for good.

- **Capture Guarantee(Postcondition):**

CG1 : Patient details data is secure to be tampered by the competitive employee.

- **Related Business Rules:**

BR1 : Admin,nurse and clerk have access to enter/delete patient data.

- **Potential Misuser Profile:**

Admin who is negligent towards his duties and lacks concentration

- **Stakeholders and Threats:**

SH1 Patient : Since its patients details that has been hacked, they are the stakeholders with most impact.
SH2 Admin: The admin could lose his/her job.
SH3 Hospital : Reputation of Hospital among its patients will bear a large negative impact, impairing the trust among the patients who have entrusted their personal details with them.

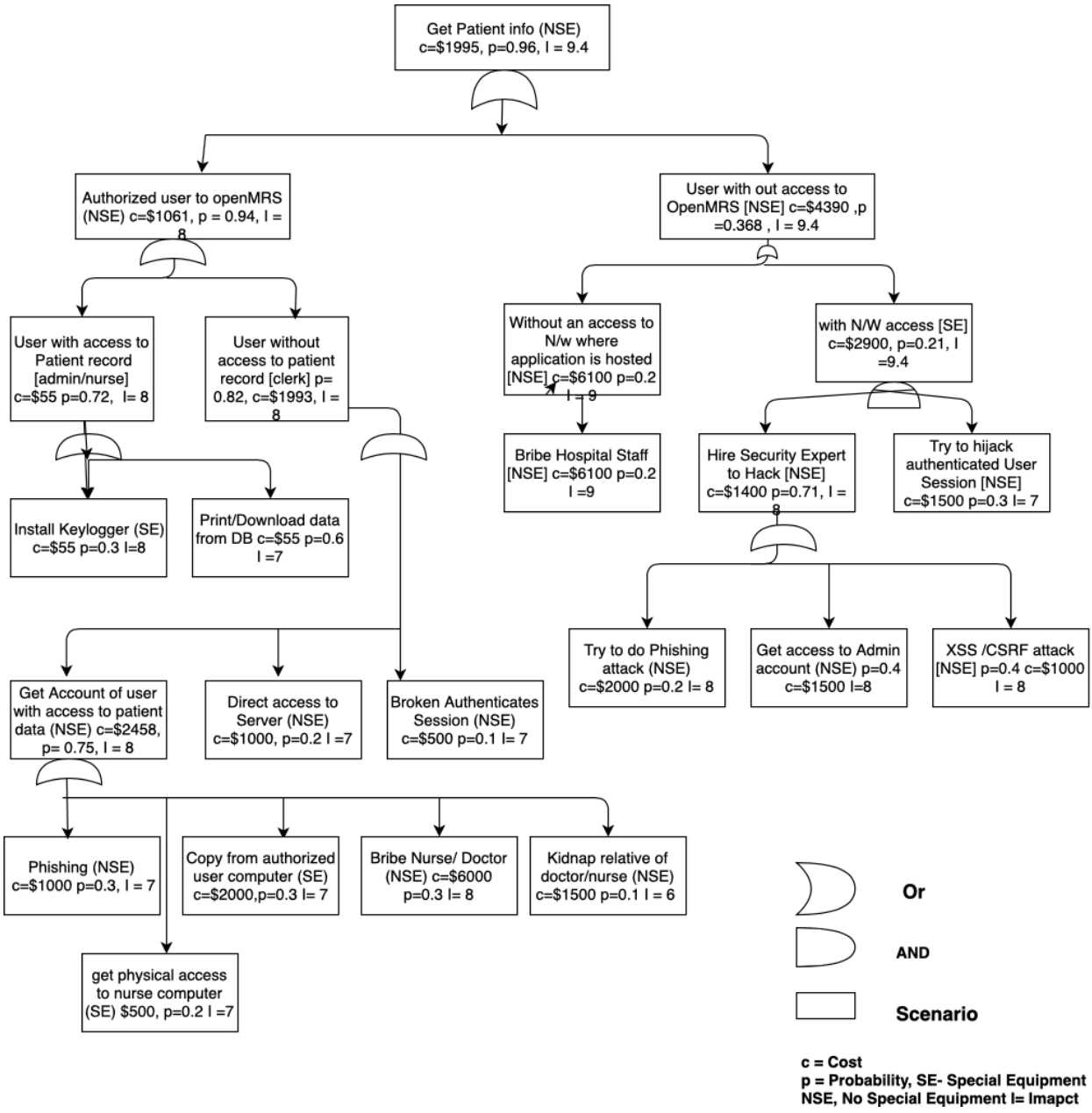
- **Scope:** Delete patient/ Start visit/Add program/Add past visit modules

- **Abstraction Level:** Misuser goal

- **Precision Level:** Focused

Solution 3:

Attack tree to steal the patient record data from Find/create patient module:



Description and Justification for Attack Tree:

The OpenMRS System can be used to attack to get the personal and Health related information from the system. These information are very critical type of information. Leaking this type of information violates Health Insurance Portability and Accountability Act (HIPAA) in United States and may violated GDPR in EU (European Union). So, protecting this sensitive information become very crucial. From the OpenMRS System we have come across the above mentioned attack tree where main aim of the tree is to steal the patient information.

These attack can be initiated either by an insider to benefit their personal motives or take revenge against their organisation(Hospital). In above tree, attack can be carried out by

someone who has authorized access to open system. This include user like Nurse/doctor which have direct access to patient record.
Second, There are users which don't have access to OpenMRS system. But, they still might need patient information. This involves Competitive hospital, Pharma companies to harm the reputation the information. They might try different method to get access the data. To carry out this they may hire some security expert to get the patient information. Each patient record cost $50 in black market and if rogue user successfully able to steal 1000 records than it sums up to $50000. Which is worth trying this attack scenario for financial gain.

Attack Scenario from a OpenMRS Authorized User:

1. The attacker in this case may be motivated to steal information for financial reasons or take revenge from the organization may be due to less hike. These patient record information can be sell in open market. These patient information can be used to do data analytics and sell their medicine records.
2. User like nurse and doctor who has access to this information may copy the personal information to a drive or print it and sell it. Or they can use keylogger in some other peers computers to put blame on someone else.
3. User like clerk who actually don't have access to the patient record can also try to use them for their personal reasons. They can get the details of other user by taking other user credential try to copy them in some media. They can try to do broken authentication ,but it is going to little difficult from someone who don't know about technology. They can also either try to kidnap or bribe their peers.

Attack Scenario from a OpenMRS UnAuthorized User:

1. In this case user will be motivated to gain financial benefit. It can be a competitive hospital which is trying to harm the reputation of the hospital or make profit to send offer to current active patient. Or it can be a new Pharma company which is trying to sell their product in new market. To study the disease pattern in the area they can try to steal the patient information.
2. As they don't have physical access to the network, so they might try to contact someone working in the hospital and try to bribe them in favor of getting patient personal health record.
3. As this become very difficult to gain access of the Database, they will hire a security expert which is very in depth knowledge about taking these kind of task. Expert somehow try to target a weak point in system by hijacking the session of any naive System user. Once, he get the access to the system then he may try to elevate his privileges by doing stored XSS or CSRF and broken authentication and hijacking the admin session. He can also try to do Phishing attack to gain the access the account of nurse and doctors.

Special Equipment required to attack on the System:

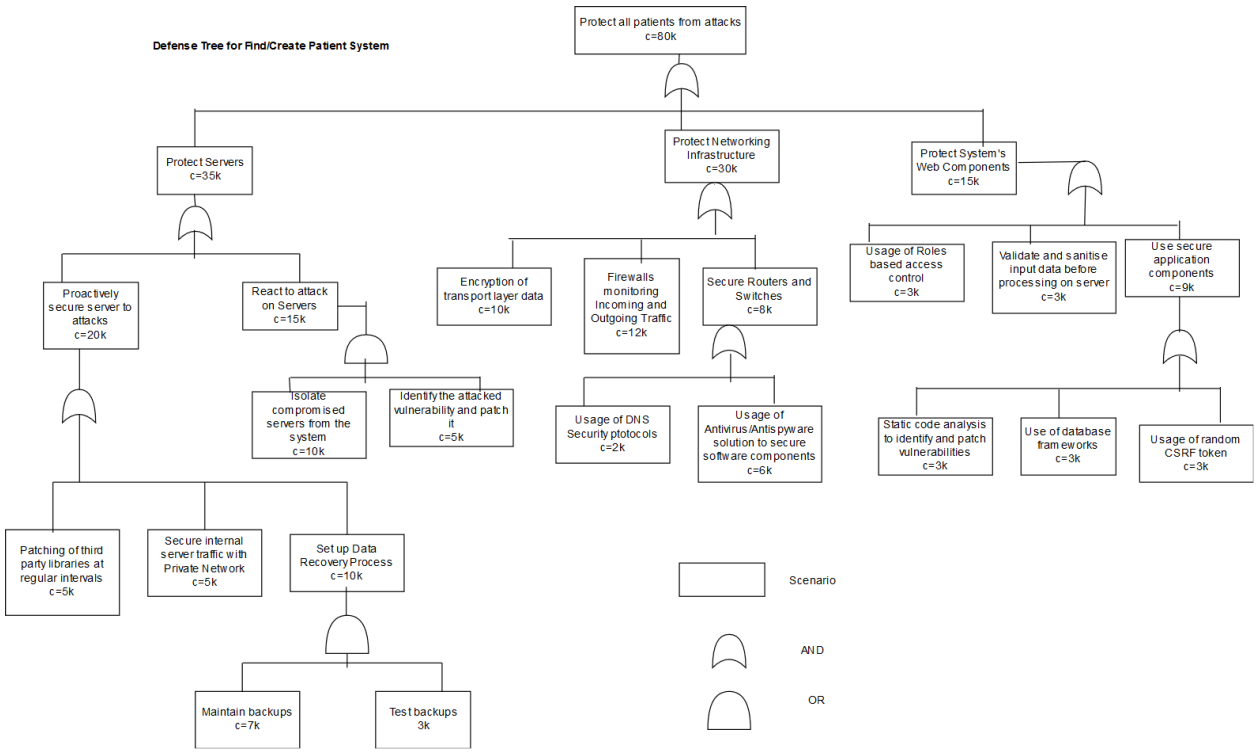
1. **Brute-force attack:** Aircrack-ng, Rainbow crack etc.

2. **Phishing attack:** LUCY, MSI Simple Phis, Gophish etc.
3. **Keylogger:** Hardware or Software based keylogger
4. **Penetration test tools:** Burp suite etc.

Citation for attack tree:

1. Kidnapping cost: https://www.therichest.com/expensive-lifestyle/it-only-costs-1-500-to-get-yourself-kidnapped/
2. Keylogger price: https://www.amazon.com/Keyllama-4MB-Value-Keylogger-black/dp/B004ZGXU48
3. Nurse can be bribe with 1 month salary: https://nursesalaryguide.net/registered-nurse-rn-salary/
4. Attack Cost: https://www.vadesecure.com/en/spear-phishing-cost/
5. https://resources.infosecinstitute.com/popular-tools-for-brute-force-attacks/#gref
6. https://www.skyhighnetworks.com/cloud-security-blog/top-phishing-test-tools-and-simulators/
7. Patient record cost in open market: https://veriphyr.com/patient-data-worth-50-each-on-black-market/

Defense tree to protect the patient record data from Find/create patient module:



Description and Logical Explanation:

OpenMRS's infrastructure,servers and web applications components should be secured in order to protect patient details from attacks.

System web application components can be secured by implementing roles based access control, which can limit the privileges to the account based on roles assigned to userid's. Also since unsanitised data can make application prone to XSS or Cross-site scripting attacks, so all the data should be sanitised before processing on server side.Futher usage of secure application components can secure the web application components. In the same context,database frameworks like Hibernate can secjure the application from SQL injection attacks.
Static code analyser can help find all the application vulnerabilities at implementation phase and prevent propagation of security bugs and flaws. Also random CSRF token can be used to secure application against CSRF(Cross-site Request Forgery) attacks.

Networking infrastructure also serves as one of the weak link for attacker and need to be secured.In the same direction certain measures can be adopted. Firstly,encryption of transport layer data can be done by SSL/TLS protocol to ensure data in transit is secure to eavesdropping. Secondly,firewalls should be configured to monitor the incoming and outgoiung network packets for suspicious activity.Thirdly,routers and switches should be secured, with use of antivirus/antispyware applications to secure it against the network intrusions of viruses and spyware,also dns security protocols like DNSSEC(DNS Security Extensions) should be used to protect dns server attacks by digitally signing data to help ensure its validity.

Servers being one of the cornerstone of OpenMRS system should be secured against the attacks by adoptiong different measures based on particular scenarios.In the first scenario of Proactive securing of server attacks, different measures can be adopted.Firstly all the third party libraries should be updated at regular intervals to ensure that security vulberabilities are addressed in the system with the latest patches released by third party provider.Secoondly, internal server traffic can be secured with private network to make it immune against the attacks by remote attackers over the network.Lastly, a data recovery proces can be setup to ensure the data is not lost after a attack and it can be rolled back to the backup copy of data , in case data is lost during attack.For same, it need to be ensured that regular data backups are maintained and also tested during the same process if data backup generated is integral and true replica of the patient data it was configured for.

Special Equipment:

1. Antivirus/Antispyware softwares like BitDefender,Norton.
2. VPC(Virtual Private Cloud) cluster for private cloud
3. Enterprise data backup solutions like Veritas
4. Static code analyser tools like Sonargraph
5. Database framework like Hibernate

Citations for defense tree:

1. DNS Security : https://www.cloudflare.com/learning/dns/dns-security/
2. VPC Cloud : https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html
3. Data backup and recovery : http://www.enterprisestorageforum.com/storage-networking/enterprise-backup-and-recovery-management.html
4. CSRF attacks security : https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet
5. Static code analysis : https://www.owasp.org/index.php/Static_Code_Analysis
6. Transport layer Security : https://hpbn.co/transport-layer-security-tls/

Solution 4:

Vulnerability History:

1) Reporting Compatibility Add On Vulnerability: CVE-2017-12796

a) The Reporting Compatibility Add On before 2.0.4 for OpenMRS, as distributed in OpenMRS Reference Application before 2.6.1, does not authenticate users when deserializing XML input into ReportSchema objects.

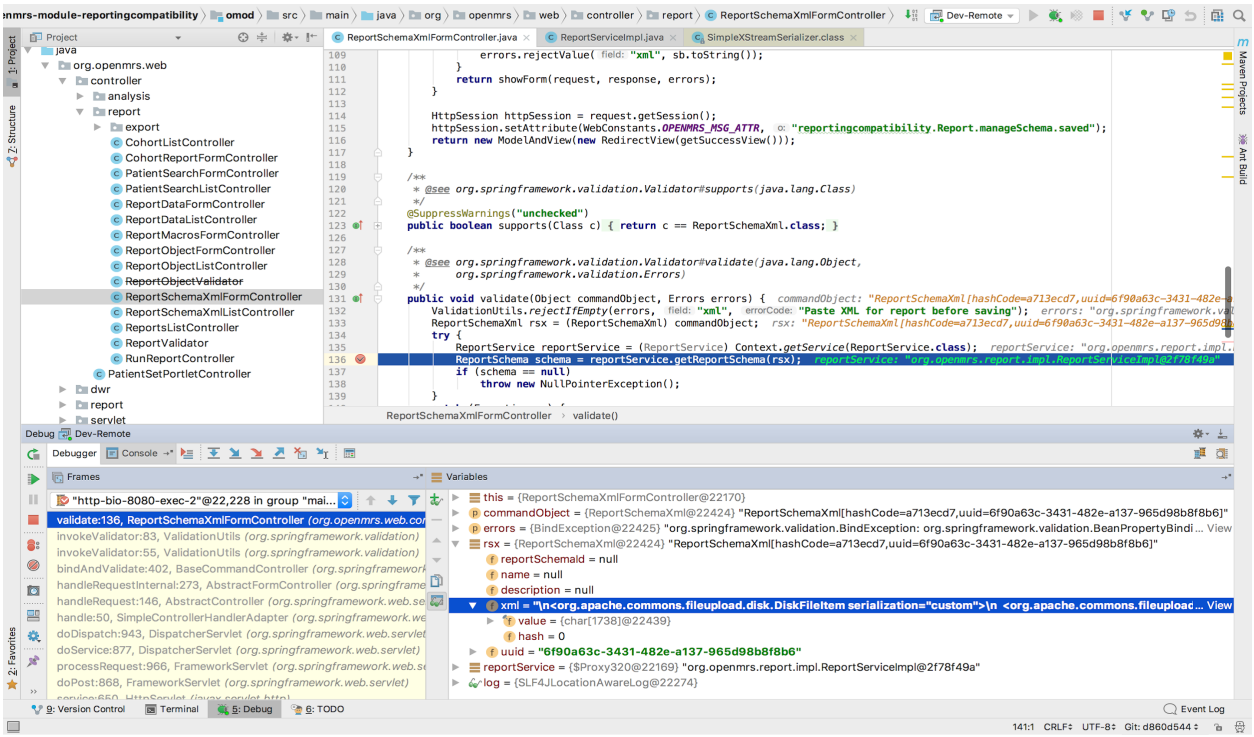
b) The result is that remote unauthenticated users are able to execute operating system commands by crafting malicious XML payloads, as demonstrated by a single admin/reports/reportSchemaXml.form request.

c) CVSS Score: 10.0

d) Exploitation of this vulnerability is possible through a single HTTP POST request to the page at http://152.46.17.224:8082/openmrs-standalone/admin/reports/reportSchemaXml.form

e) Accessing this page through a browser without authenticating first will redirect the user to the login page (so far so good). Under the hood, however, the application is actually executing server-side code before the HTTP redirect response is generated (not so good). Through a Java debugger, with a few strategically placed breakpoints, it becomes

apparent that a validation function is being called prior to any auth checks in the reportSchemaXml form controller:



f) The end result is still a HTTP 302 to the login page. The real problem here is revealed by stepping into the call to **reportService.getReportSchema(rsx),** which was identified by **Isaac Sears** and his team.'

g) This is a common recurring vulnerability CWE-502: Deserialization of Untrusted Data.

h) Mitigation Steps: Anyone running the html form entry or reporting compatibility module (included in the Reference Application) should immediately upgrade to the latest released versions of the modules, which are available on https://addons.openmrs.org/.

Reference: https://isears.github.io/jekyll/update/2017/10/21/openmrs-rce.html

2) Cross-site request forgery (CSRF): CVE-2014-8073

a) 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.

b) Exploring this vulnerability may allow an attacker to bypass certain security restrictions, obtain sensitive information, execute arbitrary script code in the browser of an unsuspecting user, steal cookie-based authentication credentials, and perform actions in the vulnerable application in the context of the victim.

```
------------------------------------------------------------------------
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>
```

c) CVSS Score: 6.8
d) This is a common recurring vulnerability CWE-352: Cross-Site Request Forgery (CSRF).
e) Mitigation Steps:
   - Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
   - Ensure that the application is free of cross-site scripting issues (CWE-79), because most CSRF defenses can be bypassed using attacker-controlled script.
   - Generate a unique nonce for each form, place the nonce into the form, and verify the nonce upon receipt of the form. Be sure that the nonce is not predictable
   - Identify especially dangerous operations. When the user performs a dangerous operation, send a separate confirmation request to ensure that the user intended to perform that operation.
   - Do not use the GET method for any request that triggers a state change.
   - Check the HTTP Referer header to see if the request originated from an expected page. This could break legitimate functionality, because users or proxies may have disabled sending the Referer for privacy reasons.

Reference: https://exchange.xforce.ibmcloud.com/vulnerabilities/97692
https://www.securityfocus.com/bid/70664
https://packetstormsecurity.com/files/128748/OpenMRS-2.1-Access-Bypass-XSS-CSRF.html

3) Administration module: CVE-2014-8072

a) The administration module in OpenMRS 2.1 Standalone Edition allows remote authenticated users to obtain read access via a direct request to /admin.
b) OpenMRS could allow a remote attacker to bypass security restrictions, caused by the failure to restrict access to the administration URL. An attacker could exploit this vulnerability to bypass security restrictions and gain access to the administration module.
c) CVSS Score: 4.0
d) This is a common recurring vulnerability CWE-264: Permissions, Privileges, and Access Controls.
e) Mitigation Steps:
    - Admins are advised to apply the appropriate updates and allow only trusted users to have network access.
    - Users are advised not to visit websites or follow links that have suspicious characteristics or cannot be verified as safe.
    - Admins are advised to use an unprivileged account when browsing the Internet and are advised to monitor critical systems.

Reference: **http://cwe.mitre.org/data/definitions/264.html**
https://exchange.xforce.ibmcloud.com/vulnerabilities/97693

4) Multiple cross-site scripting: CVE-2014-8071

a) These vulnerabilities in OpenMRS 2.1 Standalone Edition allow remote attackers to inject arbitrary web script or HTML via the (1) givenName, (2) familyName, (3) address1, or (4) address2 parameter to registrationapp/registerPatient.page; the (5) comment parameter to allergyui/allergy.page; the (6) w10 parameter to htmlformentryui/htmlform/enterHtmlForm/submit.action; the (7) HTTP Referer Header to login.htm; the (8) returnUrl parameter to htmlformentryui/htmlform/enterHtmlFormWithStandardUi.page or (9) coreapps/mergeVisits.page; or the (10) visitId parameter to htmlformentryui/htmlform/enterHtmlFormWithSimpleUi.page.
b) A remote attacker could exploit this vulnerability in a specially-crafted URL to execute script in a Web page which would be executed in a victim's Web browser within the security context of the hosting Web site, once the URL is clicked or page is viewed.
c) CVSS Score: 4.3
d) This vulnerability is usually introduced during Architecture and Design, Implementation phase.
e) This is a common recurring vulnerability CWE - 79 : Failure to Preserve Web Page Structure.
f) Mitigation Steps:
    - Use languages, libraries, or frameworks that make it easier to generate properly encoded output.

- Understand the context in which your data will be used and the encoding that will be expected.
- Use and specify a strong character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page.
- With Struts, you should write all data from form beans with the bean's filter attribute set to true.

Reference: **https://www.cvedetails.com/cwe-details/79/cwe.html**
https://www.securityfocus.com/bid/70664/info
https://exchange.xforce.ibmcloud.com/vulnerabilities/97690

Project Part 3:

0. Module Selection:

For Project Part 3, we have decided to choose "**Find/Create Patient**" module.

1. Audit/logging implementation:

The events on the portal are logged with "INFO" log levels, we will be using the same for the test cases:

1) Delete Patient:

Steps:
a) Login to OpenMRS portal through http://localhost:8082/openmrs-standalone/
Using credentials user = “admin” and password = “Admin123” and select any location.
b) Click on “Find Patient Record” and search using query “1002C4” and click on the returned patient’s record.
c) Select “Delete Patient”, when pop-up appears enter reason as “Dead” .
d) Check the log file at location “C:\openMRS\referenceapplication-standalone-2.8.0\appdata\openmrs”

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG1	Patient deleted by user event should be logged.	INFO - LoggingAdvice.invoke(115) |2018-10-26 17:15:54,972| In method VisitService.voidVisit. Arguments: Visit=Visit #376, String=Patient deleted, INFO - LoggingAdvice.invoke(155) |2018-10-26 17:15:55,004| Exiting method voidVisit INFO - LoggingAdvice.invoke(115) |2018-10-26 17:15:55,238| In method PatientService.voidPatient. Arguments: Patient=Patient#78, String=Dead, INFO - LoggingAdvice.invoke(155) |2018-10-26 17:15:55,425| Exiting method voidPatient	Adequate (Sufficient logging of patient details along with the visit details)

2) Add Allergy:

Steps:
a) Login to OpenMRS portal through http://localhost:8082/openmrs-standalone/
Using credentials user = “admin” and password = “Admin123” and select any location.
b) Click on “Find Patient Record” and search using query “10010W” and click on the returned patient’s record.
c) Click on “Add Allergy” and choose Drug as “Aspirin”, Reactions as “Cough”, Severity as “Mild” and save.
d) Check the log file at location “C:\openMRS\referenceapplication-standalone-2.8.0\appdata\openmrs”

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG2	Add Allergy record event for patient should be logged.	INFO - LoggingAdvice.invoke(115) |2018-10-26 17:30:40,970| In method UserService.saveUser. Arguments: User=admin, INFO - LoggingAdvice.invoke(155) |2018-10-26 17:30:40,970| Exiting method saveUser	Inadequate (Generic saveUser method is logged, which makes difficult to identify this event)

3) Update/Edit Allergy:

Steps:
a) Login to OpenMRS portal through http://localhost:8082/openmrs-standalone/
Using credentials user = “admin” and password = “Admin123” and select any location.
b) Click on “Find Patient Record” and search using query “10010W” and click on the returned patient’s record.
c) Click on “Edit Allergy”, edit the existing allergy, update some fields and save.
d) Check the log file at location “C:\openMRS\referenceapplication-standalone-2.8.0\appdata\openmrs”

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG 3	Update/Edit Allergy record event for patient should be logged.	INFO - LoggingAdvice.invoke(115) |2018-10-26 17:50:47,800| In method UserService.saveUser. Arguments: User=admin, INFO - LoggingAdvice.invoke(155) |2018-10-26 17:50:47,800| Exiting method saveUser	Inadequate (Generic saveUser method is logged, which makes difficult to identify this update event)

4) Remove Allergy:

Steps:
a) Login to OpenMRS portal through http://localhost:8082/openmrs-standalone/
Using credentials user = “admin” and password = “Admin123” and select any location.
b) Click on “Find Patient Record” and search using query “10010W” and click on the returned patient’s record.
c) Click on “Remove” Icon, and click “yes” on pop-up warning “Are you sure you want to remove Aspirin allergy for this patient?”.
d) Check the log file at location “C:\openMRS\referenceapplication-standalone-2.8.0\appdata\openmrs”

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG 4	Remove Allergy record event for patient should be logged.	INFO - LoggingAdvice.invoke(115) |2018-10-26 18:50:21,600| In method UserService.saveUser. Arguments: User=admin, INFO - LoggingAdvice.invoke(155) |2018-10-26 18:50:21,646| Exiting method saveUser	Inadequate (Method name logged is very vague, looks like the admin is saving something, nothing is logged about removing the allergy.

5) Start Visit Failed:

Steps:
a) Login to OpenMRS portal through http://localhost:8082/openmrs-standalone/
Using credentials user = “admin” and password = “Admin123” and select any location.
b) Click on “Find Patient Record” and search using query “10010W” and click on the returned patient’s record.
c) Click on “Start Visit”, and click “yes” on pop-up stating “Are you sure you want to start a visit for John Taylor now?”
d) Check the log file at location “C:\openMRS\referenceapplication-standalone-2.8.0\appdata\openmrs”

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG 5	Start Visit event for patient in case of error or exception should be logged.	Stack Trace for Exception Caused: Caused by: java.lang.IllegalArgumentException: Location does not support visits at org.openmrs.module.emrapi.adt.AdtServiceImpl.getLocationThatSupportsVisits(AdtServiceImpl.java:401) at org.openmrs.module.emrapi.adt.AdtServiceImpl.buildVisit(AdtServiceImpl.java:384) at org.openmrs.module.emrapi.adt.AdtServiceImpl.ensureVisit(AdtServiceImpl.java:264) at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source) at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source) at java.lang.reflect.Method.invoke(Unknown Source)	Adequate (Stack Trace helped finding the reason behind caused exception)

6) View Patient Record:

Steps:
a) Login to OpenMRS portal through http://localhost:8082/openmrs-standalone/
Using credentials user = “admin” and password = “Admin123” and select any location.
b) Click on “Find Patient Record” and search using query “10010W” and click on the returned patient’s record.
c) Check the log file at location “C:\openMRS\referenceapplication-standalone-2.8.0\appdata\openmrs”

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG 6	Viewing patient's record event should be logged including patient id, user id, timestamp.	INFO - LoggingAdvice.invoke(115) |2018-10-29 00:07:40,212| In method UserService.saveUser. Arguments: User=admin, INFO - LoggingAdvice.invoke(155) |2018-10-29 00:07:40,227| Exiting method saveUser INFO - SerializationServiceImpl.getDefaultSerializer(71) |2018-10-29 00:07:41,774| No default serializer specified - using builtin SimpleXStreamSerializer.	Inadequate (Generic logging on action which doesn't include which patient's record is viewed. It doesn't even log that the patient's record is view)

7) Register Patient:

Steps:
a) Login to OpenMRS portal through http://localhost:8082/openmrs-standalone/
Using credentials user = "admin" and password = "Admin123" and select any location.
b) Click on "Register a Patient" and enter all personal records of new patient to be registered and click on "Confirm".
c) Check the log file at location "C:\openMRS\referenceapplication-standalone-2.8.0\appdata\openmrs"

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG 7	Registering new patient event should be logged efficient, including assigned patient id.	INFO - LoggingAdvice.invoke(115) |2018-10-26 19:33:54,604| In method PatientService.savePatient. Arguments: Patient=Patient#null, INFO - LoggingAdvice.invoke(155) |2018-10-26 19:33:54,651| Exiting method savePatient	Inadequate (Logs is insufficient to identify which patient was registered)

8) Edit Patient:

Steps:
a) Login to OpenMRS portal through http://localhost:8082/openmrs-standalone/
Using credentials user = "admin" and password = "Admin123" and select any location.
b) Click on "Find Patient Record" and search using query "10010W" and click on the returned

patient's record.
c) On the top click on "Edit", change any personal information, and save form by confirming on the popped up message.
d) Check the log file at location "C:\openMRS\referenceapplication-standalone-2.8.0\appdata\openmrs"

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG 8	Edit Patient event should be logged including patient id, user, timestamp details.	INFO - LoggingAdvice.invoke(115) |2018-10-29 00:17:48,970| In method PatientService.savePatient. Arguments: Patient=Patient#37, INFO - LoggingAdvice.invoke(155) |2018-10-29 00:17:49,642| Exiting method savePatient INFO - LoggingAdvice.invoke(115) |2018-10-29 00:17:50,127| In method UserService.saveUser. Arguments: User=admin, INFO - LoggingAdvice.invoke(155) |2018-10-29 00:17:50,174| Exiting method saveUser	Adequate (Patient ID, Timestamp, user details are logged sufficiently)

9) Request Appointment:

Steps:
a) Login to OpenMRS portal through http://localhost:8082/openmrs-standalone/
Using credentials user = "admin" and password = "Admin123" and select any location.
b) Click on "Find Patient Record" and search using query "10010W" and click on the returned patient's record.
c) Click on "Request Appointment", and enter the mandatory details and save.
d) Check the log file at location "C:\openMRS\referenceapplication-standalone-2.8.0\appdata\openmrs"

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG 9	Requested Appointment event should be logged with timestamp, uuid.	INFO - LoggingAdvice.invoke(115) |2018-10-26 19:51:45,326| In method AppointmentService.saveAppointmentRequest. Arguments: AppointmentRequest=AppointmentRequest[hashCode=71a2021c,uuid=57b2e640-7363-4fff-b0ff-7cb36ba3a415], INFO - LoggingAdvice.invoke(155) |2018-10-26 19:51:45,342| Exiting method saveAppointmentRequest INFO - LoggingAdvice.invoke(115) |2018-10-26	Adequate (User details are logged along with appointment request id)

		19:51:46,990| In method UserService.saveUser. Arguments: User=admin, INFO - LoggingAdvice.invoke(155) |2018-10-26 19:51:46,990| Exiting method saveUser	

10) Merge Visits:

Steps:
a) Login to OpenMRS portal through http://localhost:8082/openmrs-standalone/
Using credentials user = “admin” and password = “Admin123” and select any location.
b) Click on “Find Patient Record” and search using query “10010W” and click on the returned patient’s record.
c) Click on “Merge Visits”, select more than 1 visits and click on “Merge Selected Visits”.
d) Check the log file at location “C:\openMRS\referenceapplication-standalone-2.8.0\appdata\openmrs”

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG10	Merge Visit event should be logged with sufficient details.	INFO - LoggingAdvice.invoke(115) |2018-10-26 19:56:23,139| In method VisitService.voidVisit. Arguments: Visit=Visit #167, String=EMR - Merge Patients: merged into visit 168, INFO - LoggingAdvice.invoke(155) |2018-10-26 19:56:23,186| Exiting method voidVisit INFO - LoggingAdvice.invoke(115) |2018-10-26 19:56:23,202| In method VisitService.saveVisit. Arguments: Visit=Visit #168, INFO - LoggingAdvice.invoke(155) |2018-10-26 19:56:23,217| Exiting method saveVisit	Adequate (Details of merged visits are logged along with the timestamp)

2. Static analysis with Fortify OR Coverity:

We have attached the pdf file for the OpenMRS run. This report is generated using Fortify tool after running on Openmrs-api.

Fortify Analysis:

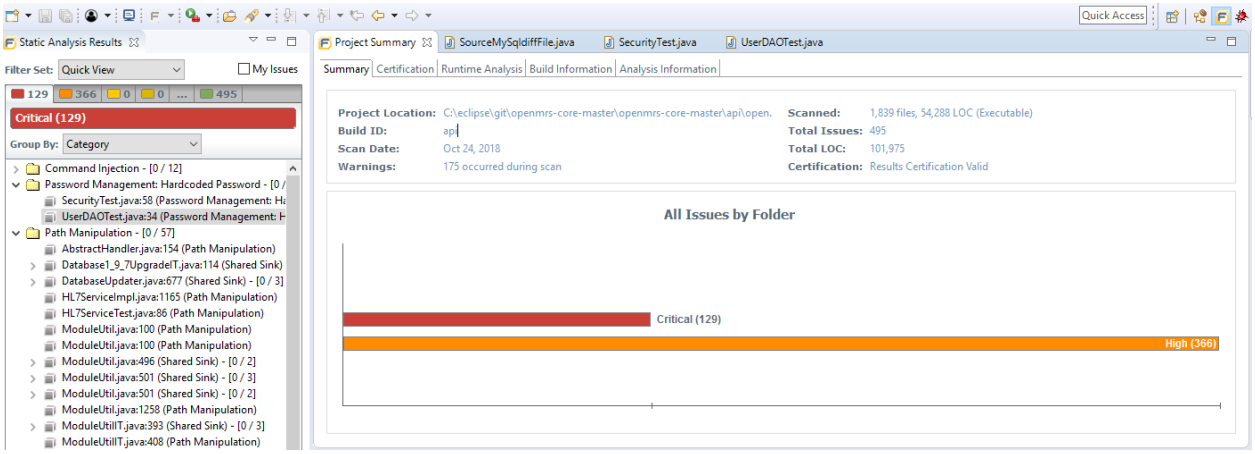


Figure: Summary for Fortify run on openmrs-api

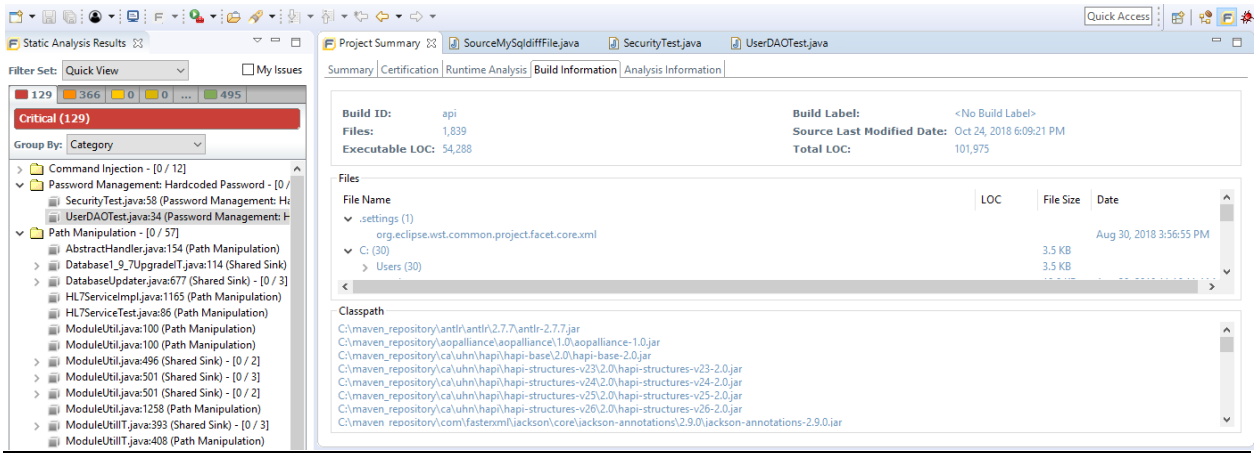


Figure: Build Information for Fortify run

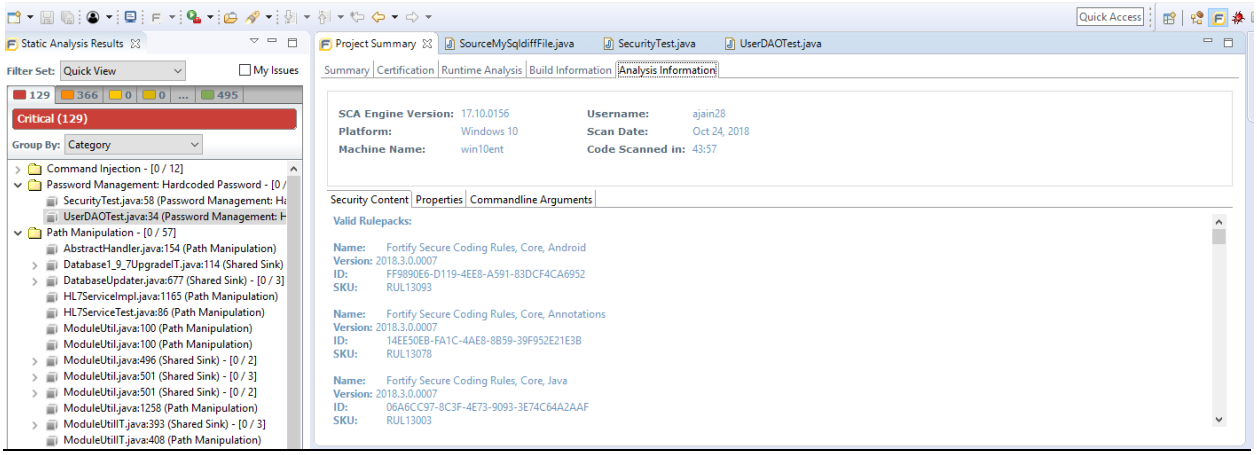


Figure: Analysis for Fortify run

Issue found in OpenMRS using Fortify tool:

1. Command Injection:

Possible Problem:

```
Process p = (wd != null) ? Runtime.getRuntime().exec(cmdWithArguments,
null, wd) : Runtime.getRuntime().exec(cmdWithArguments);
```

The method execCmd() in SourceMySqldiffFile.java at line 207 calls exec() with a command built from untrusted data. This call can cause the program to execute malicious commands on behalf of an attacker. An attacker can change the command that the program executes: the attacker explicitly controls what the command is. Or An attacker can change the environment in which the command executes: the attacker implicitly controls what the command means.

Suggested Change: To Fix this issue do not allow users to have direct control over the commands executed by the program. In cases where user input must affect the command to be run, use the input only to make a selection from a predetermined set of safe commands. If the input appears to be malicious, the value passed to the command execution function should either default to some safe selection from this set or the program should decline to execute any command at all.

Reference to Report: Details can be found at page 29 of the attached fortify report.

Weakness mitigated by change: The proposed change will try to mitigate the Command Injection.

2. Password Management: Hardcoded Password (Security Features, Structural)

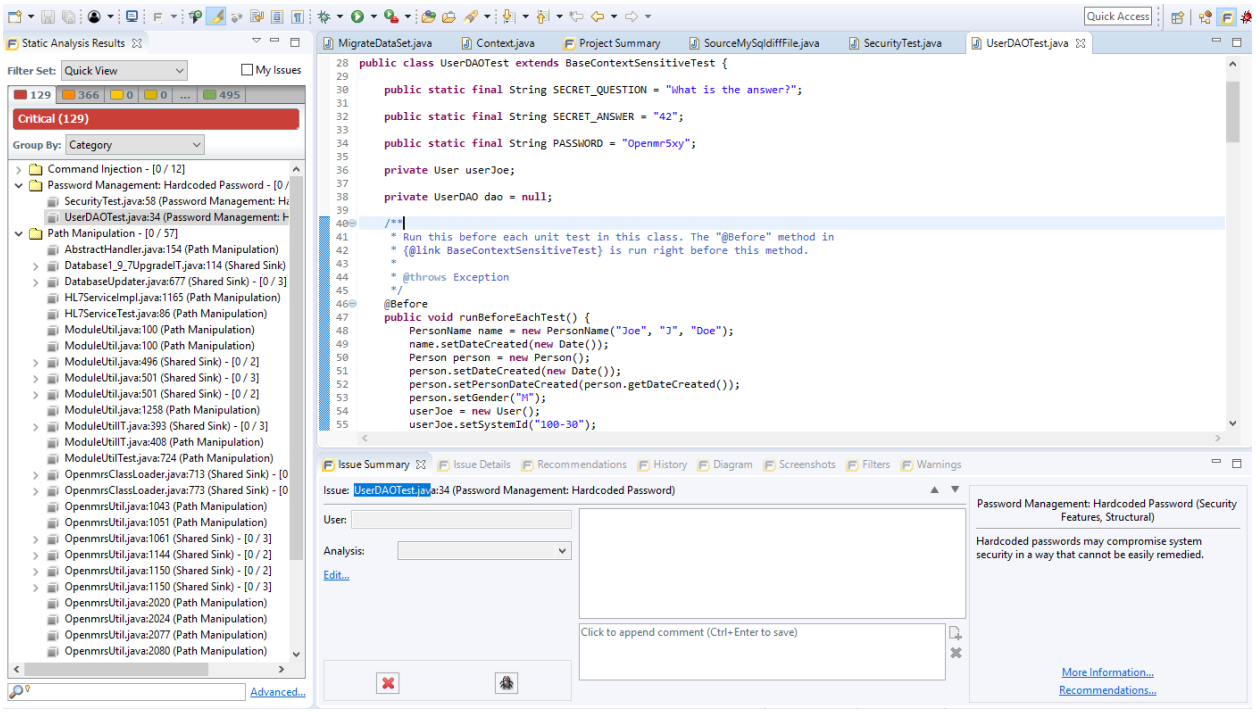
Possible Problem:
Hardcoded passwords may compromise system security in a way that cannot be easily remedied.

```
public static final String PASSWORD = "Openmr5xy";
```

It is never a good idea to hardcode a password. Not only does hardcoding a password allow all of the project's developers to view the password, it also makes fixing the problem extremely difficult. Once the code is in production, the password cannot be changed without patching the

software. If the account protected by the password is compromised, the owners of the system will be forced to choose between security and availability.

Suggested Change:
Passwords should never be hardcoded and should generally be obfuscated and managed in an external source. Storing passwords in plain text anywhere on the system allows anyone with sufficient permissions to read and potentially misuse the password. At the very least, passwords should be hashed before being stored.
Reference to report:
This vulnerability is not reported in final report, but for reference we are attaching the Image capture from our workspace.



Weakness mitigated by changes:
If we obscured the password then we can avoid Password management vulnerability.

3. Privacy Violation:

Possible Problem:
The method authenticate() in Context.java mishandles confidential information, which can compromise user privacy and is often illegal.
Privacy violations occur when Private user information enters the program. Or the data is written to an external location, such as the console, file system, or network.

In this case the data is passed to debug() in Context.java at line 293.

```
log.debug("Authenticating with username: " + username);
```

Suggested Change:
When security and privacy demands clash, privacy should usually be given the higher priority. To accomplish this and still maintain required security information, cleanse any private information before it exits the program.
We can use some internal conversion for username which is unknown to outer word and auditing purpose developer can use this information to identify the logged user.

Reference to report:
This issue can be found at page number 15 of the attached fortify report.

Weakness mitigated by changes:
The suggested change is going to mitigate privacy violation of the user.

4. Server-Side Template Injection (Input Validation and Representation, Data Flow):

Possible Problem:
The call to evaluate() in VelocityMessagePreparator.java on line 60 evaluates user-controlled data as a template engine's template, allowing attackers to access the template context and in some cases inject and run arbitrary code on the application server.
Template engines are used to render content using dynamic data. This context data is normally controlled by the user and formatted by the template to generate web pages, emails and the like. Template engines allow powerful language expressions to be used in templates in order to render dynamic content, by processing the context data with code constructs such as conditionals, loops, etc. If an attacker is able to control the template to be rendered, they will be able to inject expressions that will expose context data or even run arbitrary commands on the server.

```
engine.evaluate(context, writer, "template",template.getTemplate());
```

Suggested Change:
Whenever possible do not allow users to provide templates. If user-provided templates are necessary, perform careful input validation to prevent malicious code from being injected in the template.
In this context perform input validation for template which will try to reduce the chance of this issue.

Reference to report:

This issue can be found at page number 58 of the attached fortify report.

Weakness mitigated by changes:
The suggested change is going to mitigate Server-Side Template Injection.

5. SQL Injection (Input Validation and Representation, Data Flow)

Possible Problem:
On line 165 of MigrateAllergiesChangeSet.java, the method getConceptByGlobalProperty() invokes a SQL query built using input coming from an untrusted source. This call could allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands.
SQL injection errors occur when Data enters a program from an untrusted source. Or the data is used to dynamically construct a SQL query. In this case the data is passed to executeQuery() in MigrateAllergiesChangeSet.java at line 165.

MigrateAllergiesChangeSet.java at line 165

```
      rs = stmt.executeQuery("SELECT concept_id FROM concept WHERE uuid
= '" + uuid + "'");
```

Suggested Change:
The root cause of a SQL injection vulnerability is the ability of an attacker to change context in the SQL query, causing a value that the programmer intended to be interpreted as data to be interpreted as a command instead.
One possible solution is to use parameterized SQL statements (instead of concatenating user supplied strings) as follows:

```
...
      String uuid = ctx.getUUID();
      String query = "SELECT concept_id FROM concept WHERE uuid =?”;
      PreparedStatement stmt = conn.prepareStatement(query);
      stmt.setString(1, uuid);
      ResultSet results = stmt.execute();
...
```

Reference to report:
This issue can be found at page number 62 of the attached fortify report.

Weakness mitigated by changes:
The suggested change is going to mitigate Sql Injection by using prepared statement. We can also make use of Input validation on UUID.

6. Password Management: Hardcoded Password (Security Features, Structural)

Possible Problem:
Hardcoded passwords may compromise system security in a way that cannot be easily remedied.It is never a good idea to hardcode a password. Not only does hard coding a password allow all of the project's developers to view the password, it also makes fixing the problem extremely difficult. Once the code is in production, the password cannot be changed without patching the software. If the account protected by the password is compromised, the owners of the system will be forced to choose between security and availability.

```
public static String SCHEDULER_DEFAULT_PASSWORD = "test";
```

Suggested Change:
Passwords should never be hardcoded and should generally be obfuscated and managed in an external source. Storing passwords in plain text anywhere on the system allows anyone with sufficient permissions to read and potentially misuse the password. At the very least, passwords should be hashed before being stored.

Reference to report:
This issue can be found at page number 21 of the attached fortify report.

Weakness mitigated by changes:
The suggested change is going to mitigate this is to add this password in some sort of configuration file.

7. Key Management: Hardcoded Encryption Key (Security Features, Structural)

Possible Problem:
Hardcoded encryption keys may compromise system security in a way that cannot be easily remedied. It is never a good idea to hardcode an encryption key because it allows all of the project's developers to view the encryption key, and makes fixing the problem extremely difficult. Once the code is in production, the encryption key cannot be changed without patching the software. If the account that is protected by the encryption key is compromised, the owners of the system will be forced to choose between security and availability.

```
public static final String ENCRYPTION_KEY_SPEC = "AES";
```

Suggested Change:
Encryption keys should never be hardcoded and should be obfuscated and managed in an external source. Storing encryption keys in plain text anywhere on the system allows anyone with sufficient permissions to read and potentially misuse the encryption key.

Reference to report:
This issue can be found at page number 42 of the attached fortify report.

Weakness mitigated by changes:
This can be mitigated by storing this on some External resource. This will mitigate the security misconfiguration issue.

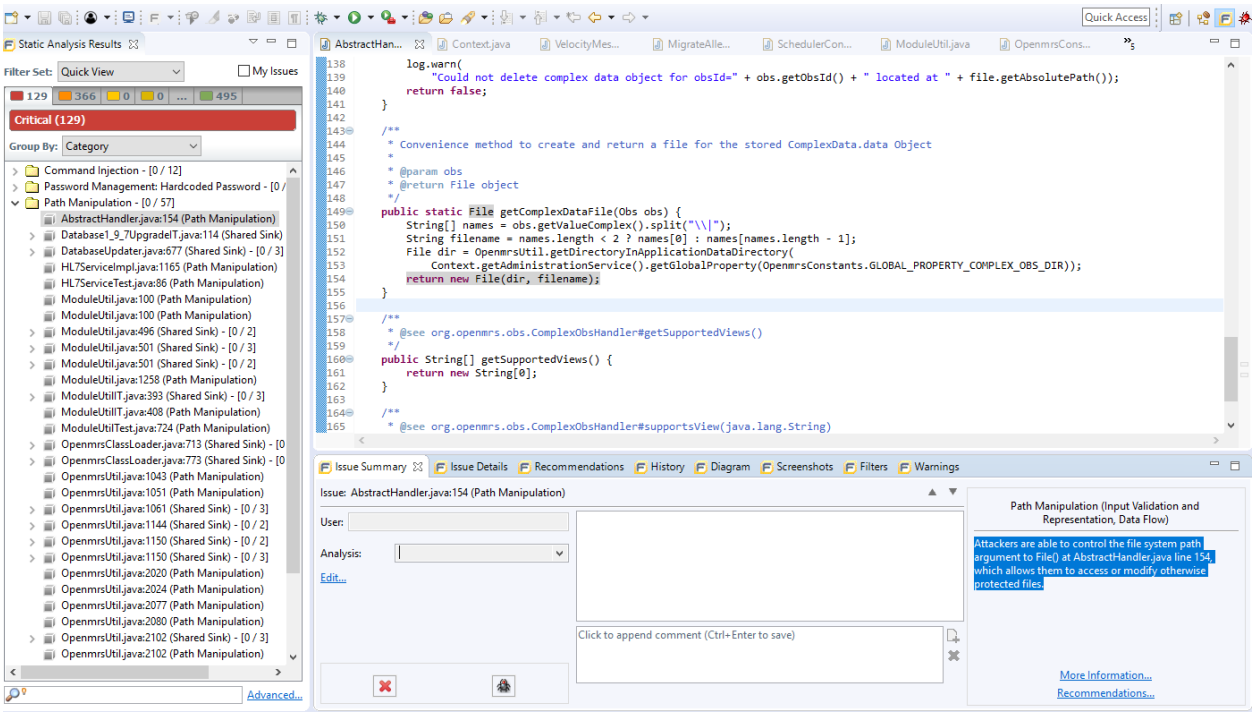
8. Path Manipulation (Input Validation and Representation, Data Flow)

Possible Problem:
Attackers are able to control the file system path argument to File() at AbstractHandler.java line 154, which allows them to access or modify otherwise protected files. Path manipulation errors occur when an attacker is able to specify a path used in an operation on the file system. Or by specifying the resource, the attacker gains a capability that would not otherwise be permitted. In this case, the attacker may specify the value that enters the program at get() in HibernateObsDAO.java at line 73, and this value is used to access a file system resource at File() in AbstractHandler.java at line 154.

```
return new File(dir, filename);
```

Suggested Change:
The best way to prevent path manipulation is with a level of indirection: create a list of legitimate resource names that a user is allowed to specify, and only allow the user to select from the list. With this approach the input provided by the user is never used directly to specify the resource name.

Reference to report:
This vulnerability is not reported in final report, but for reference we are attaching the Image capture from our workspace.



Weakness mitigated by changes:
This can mitigate the Indirect object reference.

9. Denial of Service: Regular Expression (Input Validation and Representation, Data Flow)

Possible Problem:
Untrusted data is passed to the application and used as a regular expression. This can cause the thread to over consume CPU resources.There is a vulnerability in implementations of regular expression evaluators and related methods that can cause the thread to hang when evaluating regular expressions that contain a grouping expression that is itself repeated. Additionally, any regular expression that contains alternate subexpressions that overlap one another can also be exploited. This defect can be used to execute a Denial of Service (DoS) attack.

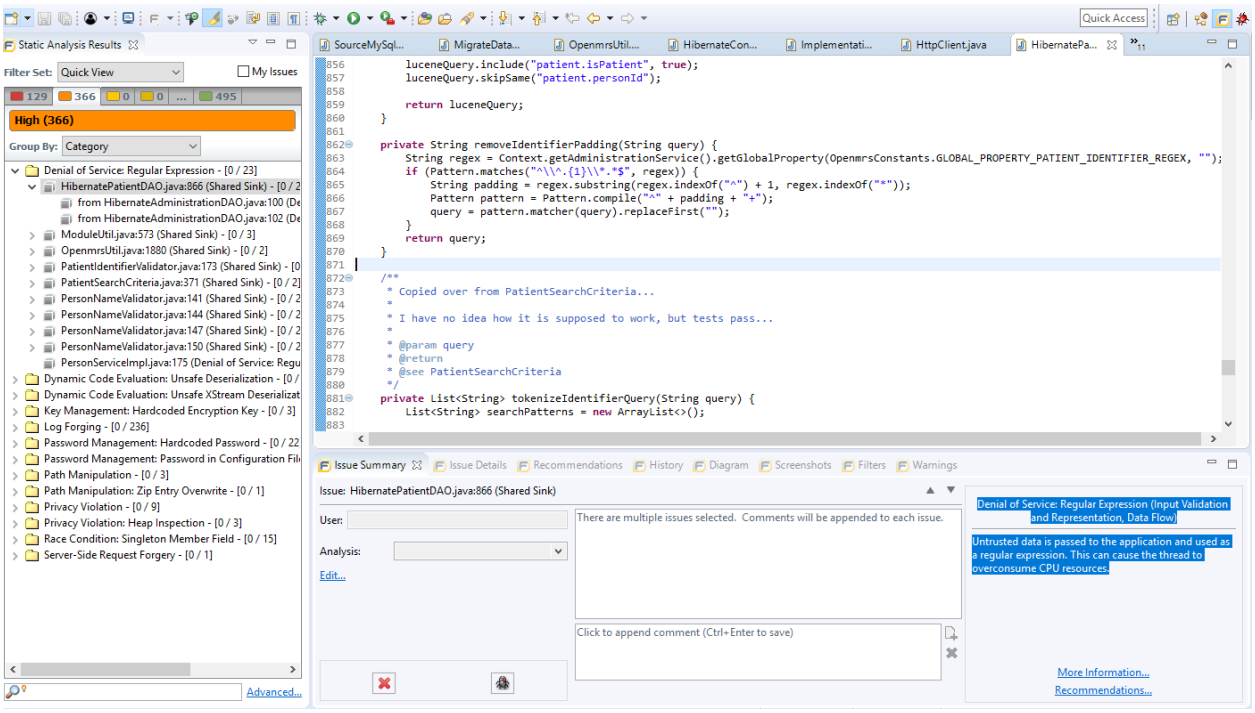
```
                Pattern pattern = Pattern.compile("^" + padding +
"+");

                query = pattern.matcher(query).replaceFirst("");
```

Suggested Change:

Do not allow untrusted data to be used as regular expression patterns.

Reference to report:
This vulnerability is not reported in final report, but for reference we are attaching the Image capture from our workspace.



Weakness mitigated by changes:
This issue handle DDOS attack.

10. Server-Side Request Forgery (Input Validation and Representation, Data Flow)

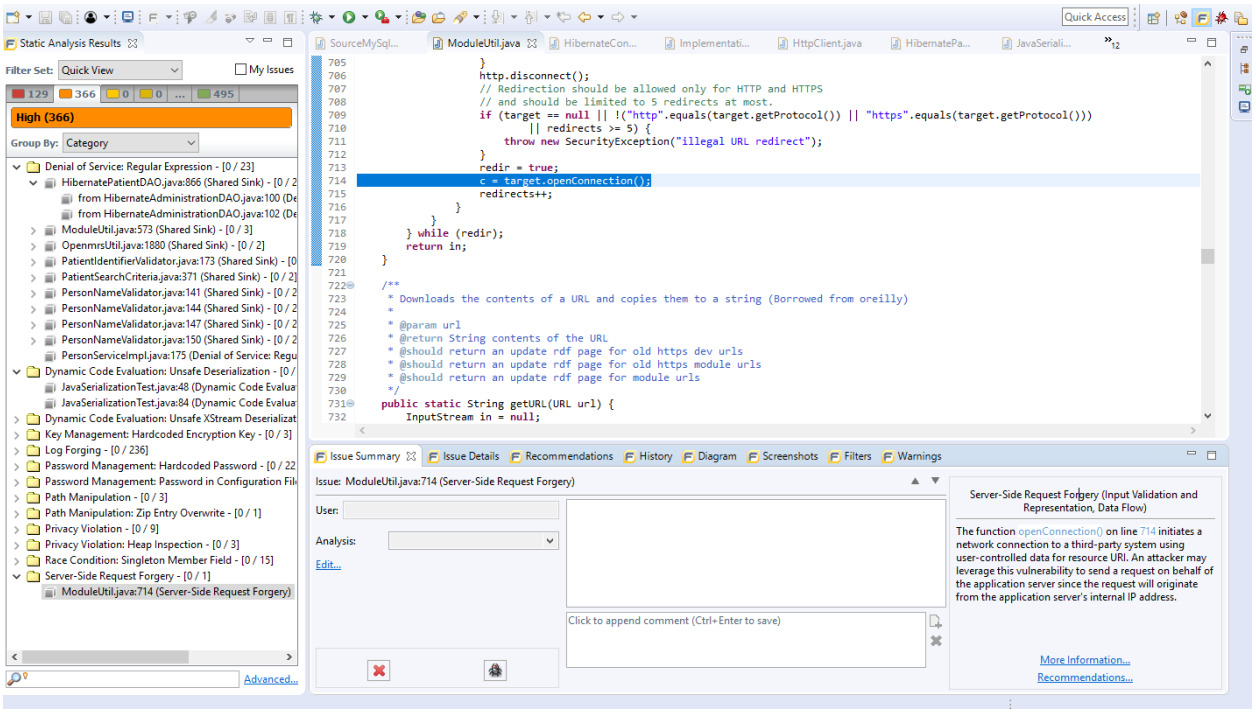
Possible Problem:
The function openConnection() on line 714 initiates a network connection to a third-party system using user-controlled data for resource URI. An attacker may leverage this vulnerability to send a request on behalf of the application server since the request will originate from the application server's internal IP address. A Server-Side Request Forgery occurs when an attacker may influence a network connection made by the application server.

```
c = target.openConnection();
```

Suggested Change:

Do not establish network connections based on user-controlled data and ensure that the request is being sent to the expected destination. If user data is necessary to build the destination URI, use a level of indirection: create a list of legitimate resource names that a user is allowed to specify, and only allow the user to select from the list. With this approach the input provided by the user is never used directly to specify the resource name.

Reference to report:
This vulnerability is not reported in final report, but for reference we are attaching the Image capture from our workspace.



Weakness mitigated by changes:
If we mitigate this issue then we can handle the Server-side Request Forgery request.

Issue found in OpenMRS using Coverity tool:

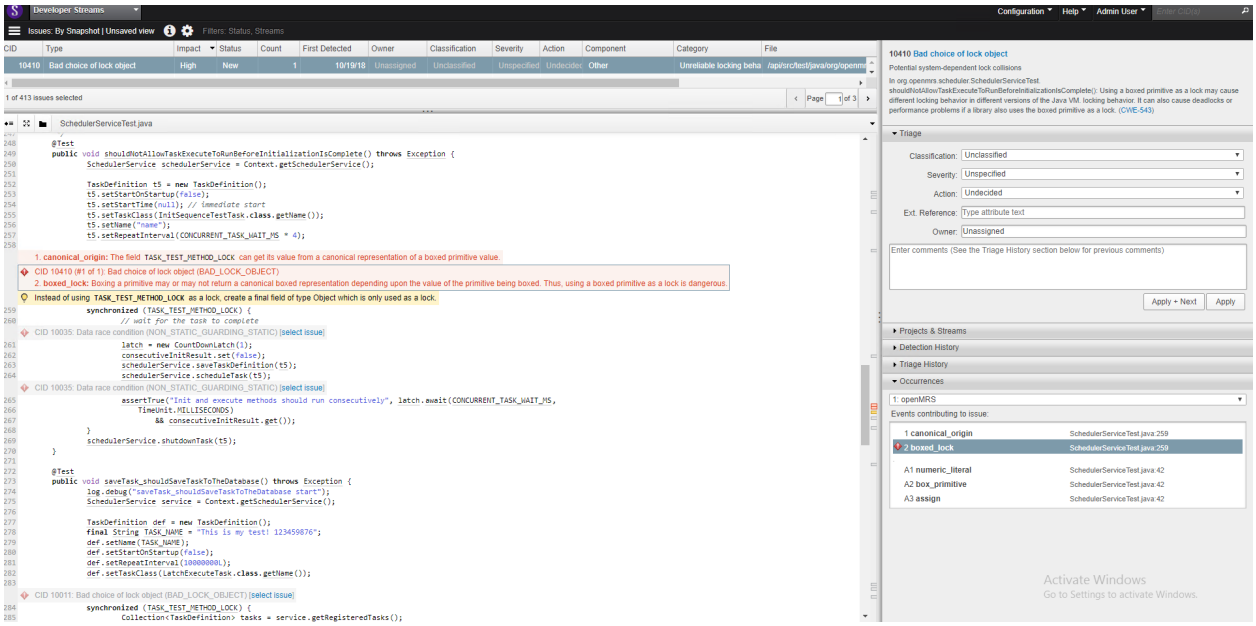
1. Bad choice of lock object

Problem Description:
It happens in org.openmrs.scheduler.SchedulerServiceTest.
shouldNotAllowTaskExecuteToRunBeforeInitializationIsComplete(): Here a boxed primitive as a lock is used which shsows in variance in locking behaviour across different

versions of Java VM, and it can also cause deadlocks or performance problems if a library also uses the boxed primitive as a lock. (CWE-543)

Suggested change:
Variable TASK_TEST_METHOD_LOCK is a boxed primitive which can cause deadlock with libraries using boxed primitve as alock.So instead of that a final variable of object type can be used for lock.

Reference to report:



Weakness mitigated by changes:
Changes suggested will help preventing the deadlocks and varying locking behaviour across Java VM's.

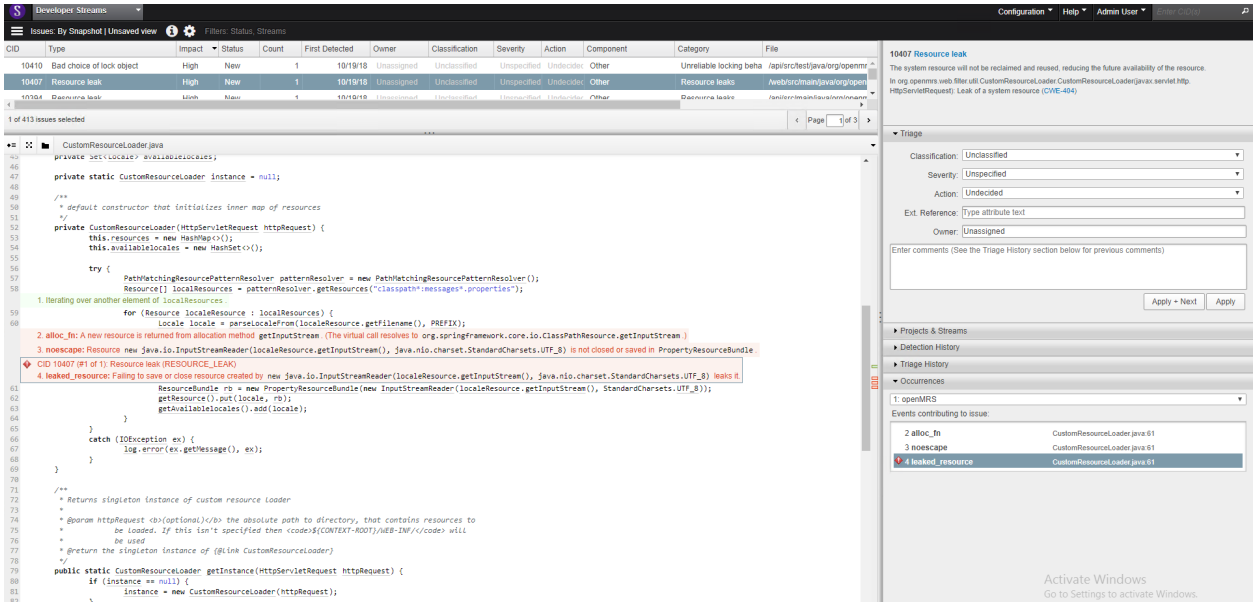
2. Resource leak

Problem Description:
It happens in org.openmrs.web.filter.util.CustomResourceLoader.
CustomResourceLoader(javax.servlet.http.HttpServletRequest)

So in that file, system resource was not reclaimed and reused, reducing the future availability of the resource.Leak of a system resource (CWE-404)

Suggested change:
InputStream reference should be closed by calling close() method on it inside a new finally block which can be added to release resources both in the normal and exception flow.

Reference to report:



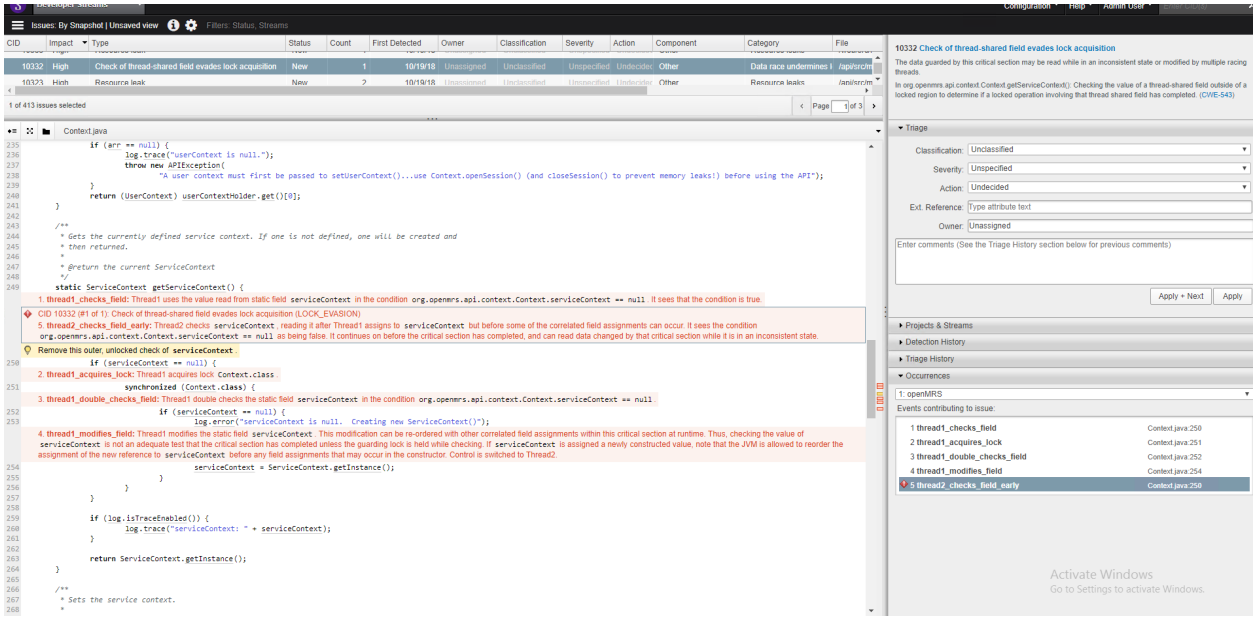
Weakness mitigated by changes:
Changes will help reducing the fragmentation of system resources, and will help reclaiming the unused system resources.

3. Check of thread-shared field evades lock acquisition

Problem Description:
It happens in org.openmrs.api.context.Context.getServiceContext()
Here value of a thread-shared field outside of a locked region is checked to determine if a locked operation involving that thread shared field has completed. (CWE-543) which may result in the data guarded by this critical section may be read while in an inconsistent state or modified by multiple racing threads.

Suggested change:
Remove double-check locking pattern and use a synchronised call instead to prevent the race condition.

Reference to report:



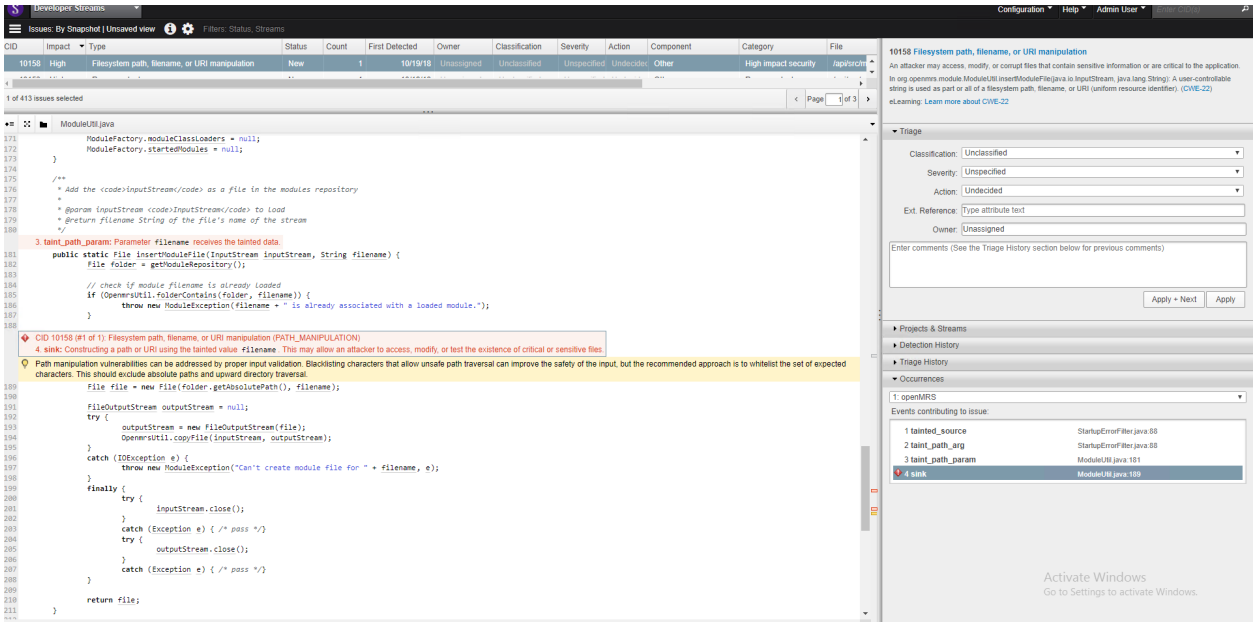
Weakness mitigated by changes:
Suggested change will mitigate the race condition as exhibited above, and will help prevent evasion of lock acquisition.

4. Filesystem path, filename, or URI manipulation

Problem Description:
It happens in org.openmrs.module.ModuleUtil.insertModuleFile(java.io.InputStream, java.lang.String)
So here a user-controllable string is used as part or all of a filesystem path, filename, or URI (uniform resource identifier). (CWE-22), because of which an attacker may access, modify, or corrupt files that contain sensitive information or are critical to the application.

Suggested change:
Here firstly input should be validated, secondly all the expected characters should be whitelisted to exclude absolute path and upward directory traversal.

Reference to report :



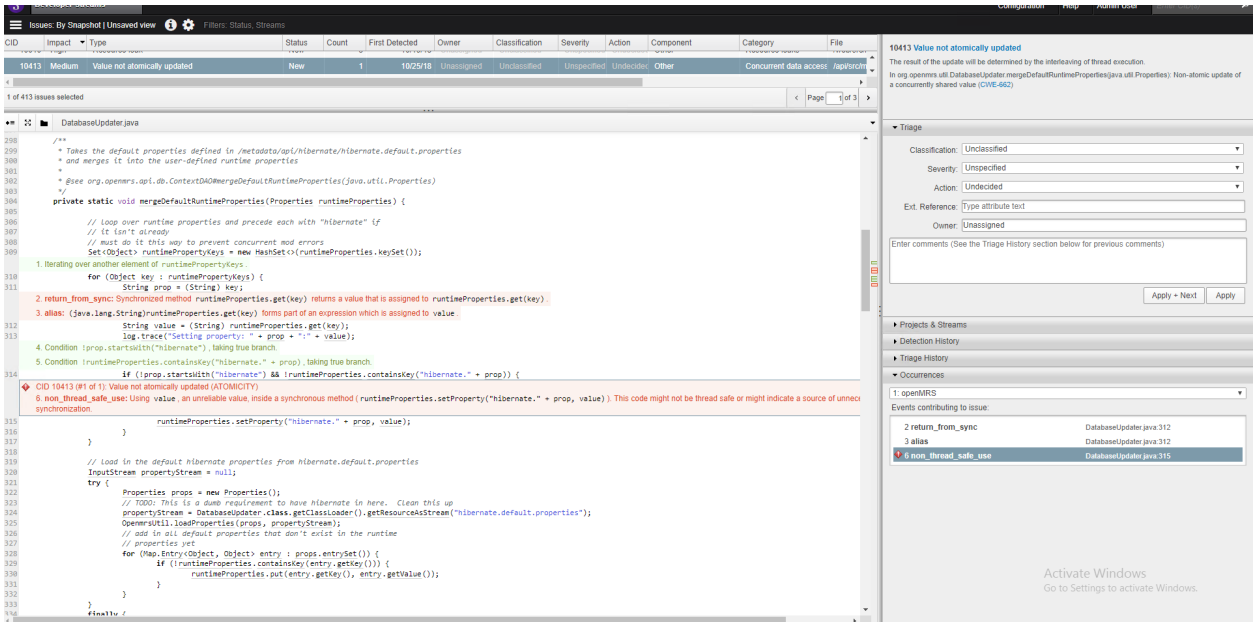
Weakness mitigated by changes:
Suggested changes will help in excluding absolute paths and upward directory traversal,thus mitigating malicious manipulation of filename or path.

5. Value not atomically updated

Problem Description:
It happens in org.openmrs.util.DatabaseUpdater.mergeDefaultRuntimeProperties(java.util.Properties)
Here non-atomic update of a concurrently shared value (CWE-662) is performed, with result of the update determined by the interleaving of thread execution.

Suggested change:
Code can be made thread-safe by using synchronising the transactions with synchronise method provided by java api.

Reference to report:



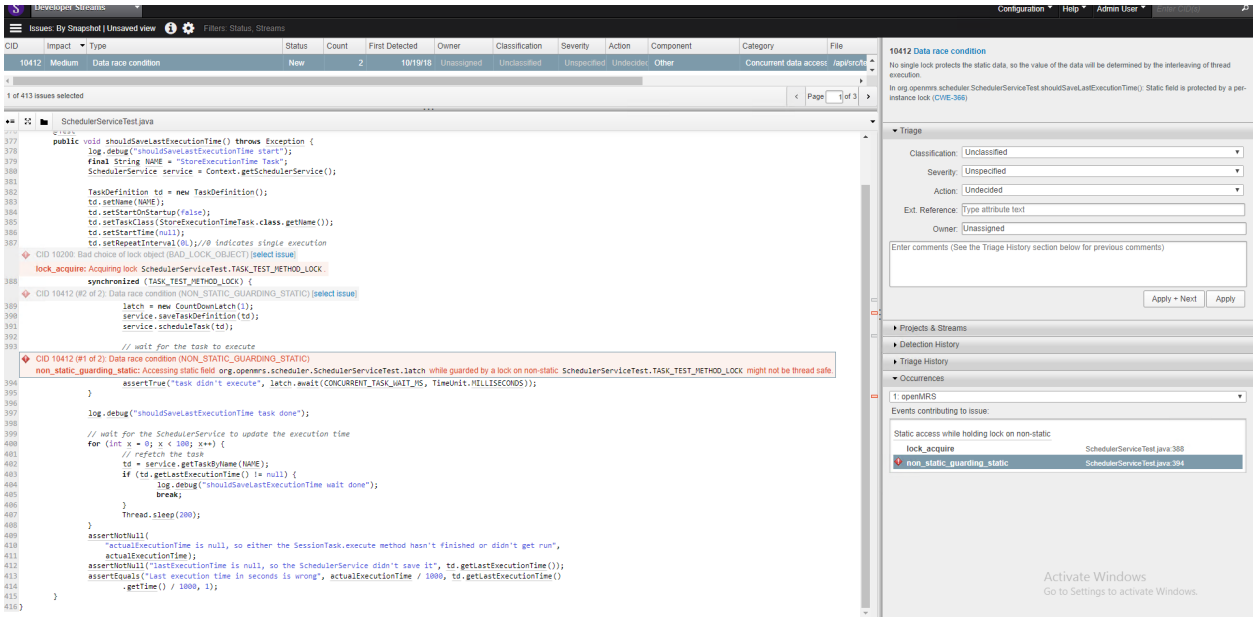
Weakness mitigated by changes:
Unreliable updates can be mitigated,by synchronising the transactions in atomic manner.

6. Data race condition

Problem Description:
It happened in org.openmrs.scheduler.SchedulerServiceTest. shouldSaveLastExecutionTime().Here Static field is protected by a per-instance lock (CWE-366) with no single lock protects the static data, so the value of the data will be determined by the interleaving of thread execution.

Suggested change:
Implement locking mechanism around the code involved in alter or read operations of data in multithreaded environment. Also resource-locking sanity checks can be implemented to enforce a blocking scheme where resources are being used by other threads of execution.

Reference to report:



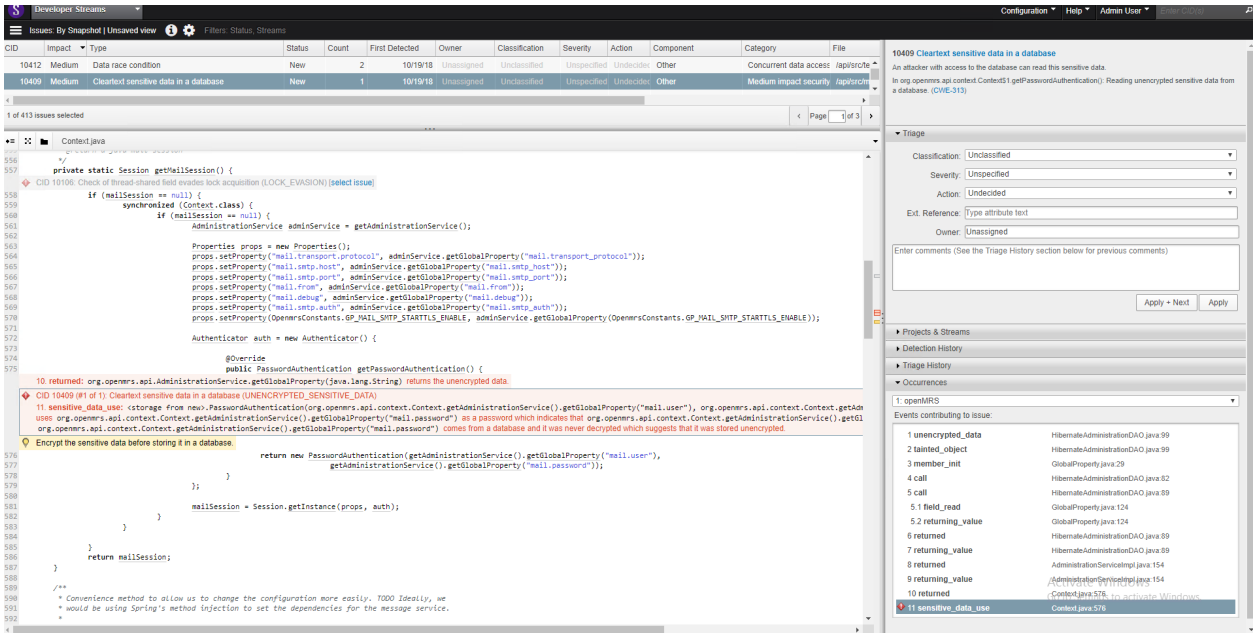
Weakness mitigated by changes:
Suggested changes will help mitigate the resource locking issues in case of interleaving thread execution.

7. Cleartext sensitive data in a database

Problem Description:
It happens org.openmrs.api.context.Context$1.getPasswordAuthentication() with read of unencrypted sensitive data from a database. (CWE-313) due to which an attacker with access to the database can read this sensitive data.

Suggested change:
Sensitive data in this issue can be encrypted before storing it in database.

Reference to report:



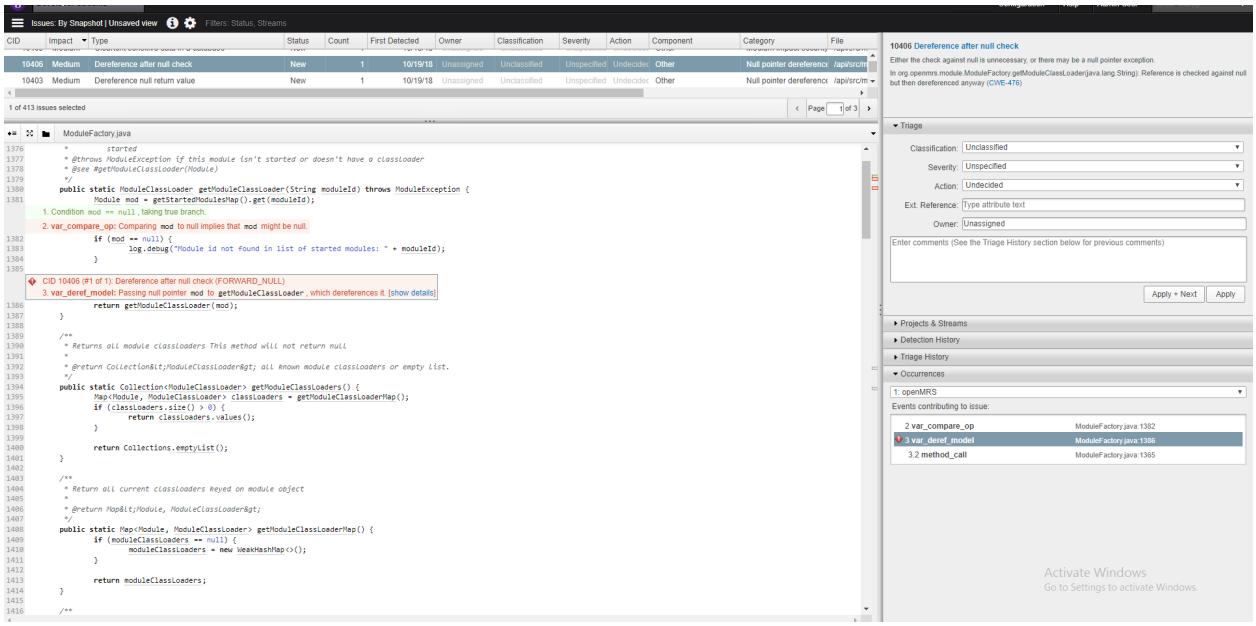
Weakness mitigated by changes:
Changes will help prevent the revelation of sensitive data in case of attacks done on database.

8. Dereference after null check

Problem Description:
It happens in org.openmrs.module.ModuleFactory.getModuleClassLoader(java.lang. String) where reference is checked against null but then dereferenced anyway (CWE-476) which may cause a null pointer exception.

Suggested change:
Check the value is non-null before returning or acting upon it.

Reference to report:



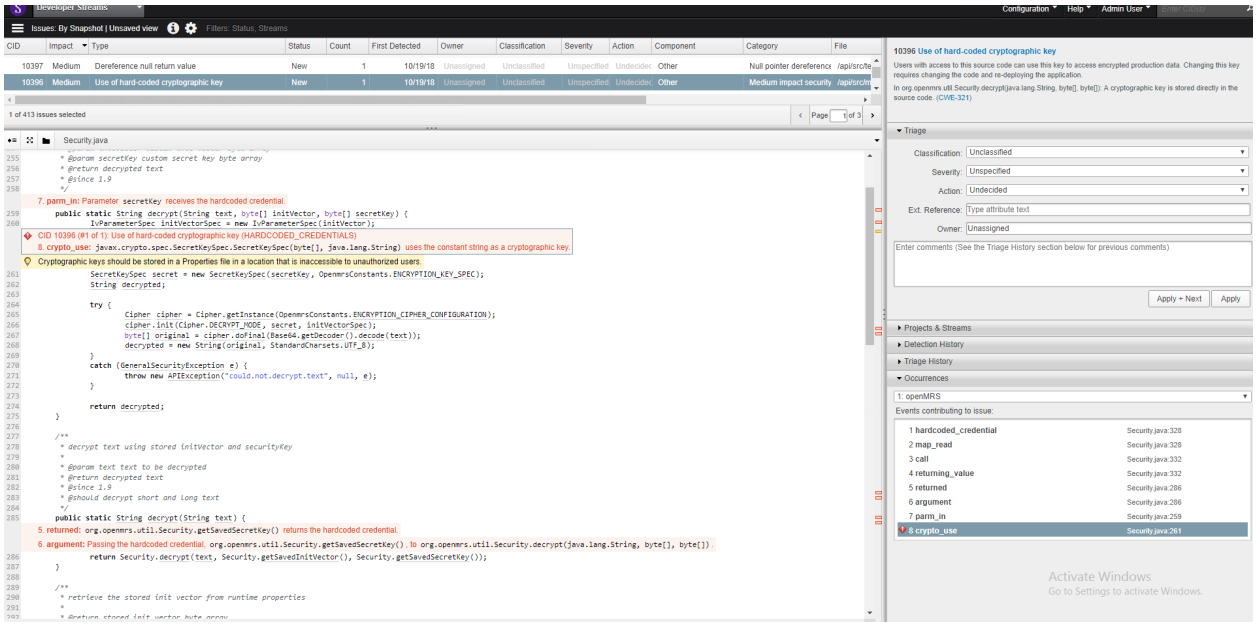
Weakness mitigated by changes:
Runtime exceptions like null pointer exceptions can be handled and dealt with in a graceful manner ,rather than presenting an abrupt exception.

9. Use of hard coded cryptographic key

Problem Description:
It happens in org.openmrs.util.Security.decrypt(java.lang.String, byte[], byte[]) with a cryptographic key is stored directly in the source code. (CWE-321).Here users with access to this source code can use this key to access encrypted production data and changing this key requires changing the code and re-deploying the application.

Suggested change:
Cryptographic keys should not be hardcoded and instead should be stored in a properties file,which is inaccessible to unauthorized users.

Reference to report:



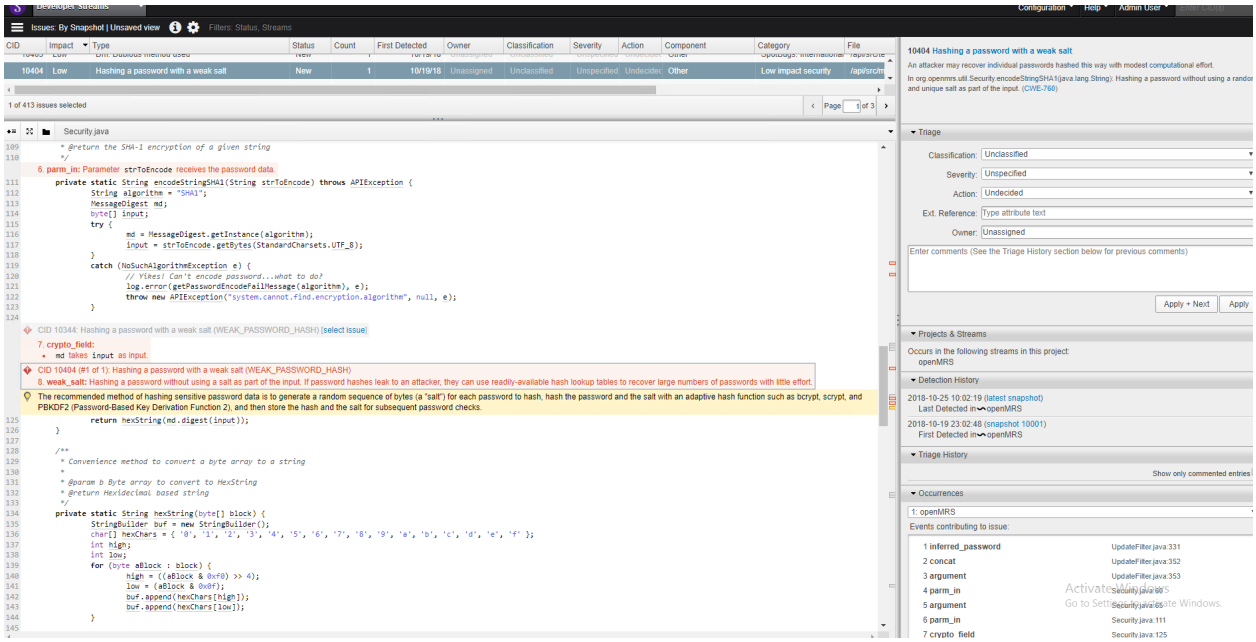
Weakness mitigated by changes:
Suggested changes will help preventing revelation of cryptographic keys to people with access to source code, which can prevent misuse of those keys.

10. Hashing a password with a weak salt

Problem Description:
It happens in org.openmrs.util.Security.encodeStringSHA1(java.lang.String) with hashing of password done,without using a random and unique salt as part of the input. (CWE-760).n attacker may recover individual passwords hashed this way with modest computational effort.

Suggested change:
A random sequence of bytes or hash should be generated for each password to hash and then storage of password and hash for subsequent password checks.

Reference to report:



Weakness mitigated by changes:
Suggested changes can mitigate the dictionary,brute-force and rainbow table attacks for recovering the password.

3. Fuzzing with ZAP:

Test Case ID	Project3_JBroFuzz_1
Test Name	Running JBro Fuzzing tool to identify XSS on create patient
Fuzzer Chosen	XSS: (RuleSet: XSS 101)
Steps to reproduce	1. Setup the ZAP proxy tool as described on this link. ZAP tutorial 2. Use Section to set a breakpoint and ZAP fuzzing from it. 3. Run the Open MRS application on the localhost. 4. Goto the URL: http://localhost:8082/openmrs-standalone/login.htm 5. Login as Admin. Username: admin, password: “Admin123”, select Impatient ward and hit login. 6. Once you are login into OpenMRS. Go to the Register Patient tab in the menu and select it. 7. If you haven’t Open ZAP proxy, select the icon from the desktop and opened it. Till now you would have followed the step 1 which has done setup of proxy. 8. Create a patient with entering details as request on the screen. This can be done by

	filling first name, Middle Name, Family name & hit enter after entering the Family Name. 9. Now, go to ZAP proxy. And Navigate to openmrs-standalone -> registrationapp -> matchingpatient. There you will see some post request. Select one of the post request and select the Request tab from right hand side. After you are done with this Steps your screen will look something similar like Figure 1. 10. Now Select a field value from the Request value pane from the rightmost Down. We have selected **preferred field** value and right click on the selected value. And select the Fuzz from Menu option. 11. Select Payload from Dialog Box and click on Add. 12. From the Drop down of Add Payload select “File Fuzzers” and expand on Jbrofuzz and then look for XSS and expand it also. Select XSS101 form the Ruleset. 13. And click on add button and then OK button and later Start the fuzzer.
Result	Tried out all the probable combination of XSS string and didn’t find any vulnerability.
Vulnerability found	No
Reason for result	Validation is implemented in the OpenMRS source code to handle the XSS attack.
Adjustment to fuzzing rules	Tried other combination of XSS by selecting all the probable XSS string and we see for some of the issue it shows reflected XSS but they return 500 HTTP code.
Mitigation Steps	N/A

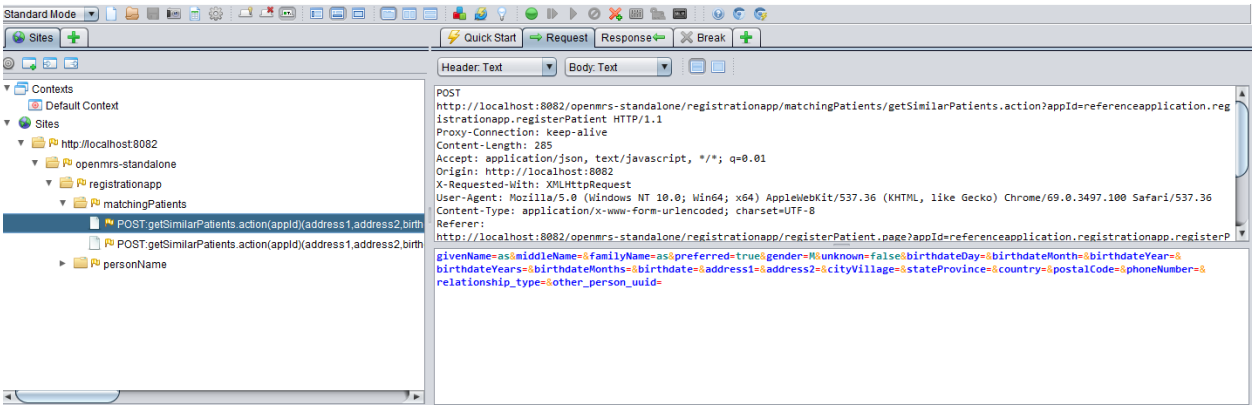


Figure 1: Post request for Jbro fuzz tool

Test Case ID	Project3_JBroFuzz_2
Test Name	Running JBro Fuzzing tool to identify Sql Injection on Create Patient
Fuzzer Chosen	SQL Injection (RuleSet: SQL Injection 101)

Steps to reproduce	1. Setup the ZAP proxy tool as described on this link. ZAP tutorial 2. Use Section to set a breakpoint and ZAP fuzzing from it. 3. Run the Open MRS application on the localhost. 4. Goto the URL: http://localhost:8082/openmrs-standalone/login.htm 5. Login as Admin. Username: admin, password: “Admin123”, select Impatient ward and hit login. 6. Once you are login into OpenMRS. Go to the Register Patient tab in the menu and select it. 7. If you haven’t Open ZAP proxy, select the icon from the desktop and opened it. Till now you would have followed the step 1 which has done setup of proxy. 8. Create a patient with entering details as request on the screen. This can be done by filling first name, Middle Name, Family name and hit enter after entering the Family Name. 9. Now, go to ZAP proxy. And Navigate to openmrs-standalone->registrationapp->matchingpatient. There you will see some post request. Select one of the post request and select the Request tab from right hand side. After you are done with this Steps your screen will look something similar like Figure 1. 10. Now Select a field value from the Request value pane from the rightmost Down. We have selected **family Name** value and right click on the selected value. And select the Fuzz from Menu option. 11. Select Payload from Dialog Box and click on Add. 12. From the Drop down of Add Payload select “File Fuzzers” and expand on Jbrofuzz and then look for SQL Injection and expand it also. Select MySQL Injection 101 . OpenMRS uses Mysql db. 13. And click on add button and then OK button and later Start the fuzzer.
Result	Tried all the possible sql attack strings and fuzzer didn’t find any vulnerability.
Vulnerability found	No
Reason for result	OpenMRS might be using parameterized query or doing the server side validation of these input.
Adjustment to fuzzing rules	Select all possible Sql injection string from the list of fuzzer and try to run the fuzzer. Fuzzer don’t show us any sign of vulnerability on this.
Mitigation Steps	N/A

Test Case ID	**Project3_JBroFuzz_3**
Test Name	Running JBro Fuzzing tool to identify Buffer overflow on Edit Patient
Fuzzer Chosen	Buffer Overflow (RuleSet : Long Strings of aaa’s)

Steps to reproduce	1. Setup the ZAP proxy tool as described on this link. ZAP tutorial and also open the ZAP tool and configure local proxy as needed. 2. Now in that doc, follow the steps listed under section to set a breakpoint and ZAP fuzzing from it. 3. Run the Open MRS application on the localhost. 4. Goto the URL: http://localhost:8082/openmrs-standalone/login.htm 5. Login as Admin. Username: admin, password: “Admin123”, select Isolation ward and hit login. 6. Once you are login into OpenMRS. Go to the Find Patient Record tab in the menu and then on new screen,enter John in the search box and hit enter. 7. Now in the search results that appears,select the row with record of John Smith and click on it. 8. Now patient details page appears,then click on Edit button. 9. Now on the page that appears,change some details there and click on Save Form and then click on Confirm button. 10. Go to ZAP and locate POST request for it, which in this case is POST:editSection 11. Now go to the Request tab,and highlight the familyname value there and right click to select and open Fuzz. 12. Now click on Payloads button and then click on Add button. 13. Now select File Fuzzers from the Type dropdown and then expand the jbrofuzz and then select Buffer overflow checkbox listed under it and then click on Add button and then click ok and then click on Start Fuzzer.
Result	Tried all the accepted patterns,but didn’t found out vulnerabilities.
Vulnerability found	No
Reason for result	Validations had been implemented in openmrs for checking maximum field length.
Adjustment to fuzzing rules	Will try the long list of numbers also with fields of numerical value.
Mitigation Steps	N/A

Test Case ID	**Project3_JBroFuzz_4**
Test Name	Running JBro Fuzzing tool to identify Injection on Edit Patient
Fuzzer Chosen	Injection(RuleSet : MySQL Injection 101)
Steps to reproduce	14. Setup the ZAP proxy tool as described on this link. ZAP tutorial and also open the ZAP tool and configure local proxy as needed. 15. Now in that doc, follow the steps listed under section to set a breakpoint and ZAP fuzzing from it.

	16. Run the Open MRS application on the localhost. 17. Goto the URL: http://localhost:8082/openmrs-standalone/login.htm 18. Login as Admin. Username: admin, password: “Admin123”, select Isolation ward and hit login. 19. Once you are login into OpenMRS. Go to the Find/Create Patient page at http://localhost:8082/openmrs-standalone/findPatient.htm.. 20. Now in the Find Patient section enter name as ‘John’ and click enter. 21. Go to ZAP and locate POST request for it, which in this case is POST:DWRPatientService.findCountAndPatients.dwr(callCount) 22. Now go to the Request tab,and highlight the value John stored in c0-param0=string variable,then right click to select and open Fuzz. 23. Now click on Payloads button and then click on Add button. 24. Now select File Fuzzers from the Type dropdown and then expand the jbrofuzz and then select Injection checkbox listed under it and then select MySQL Injection 101 ,and click on Add button and then click ok and then click on Start Fuzzer.
Result	No vulnerabilities found, for all the patterns listed in ruleset of MySQL injection 101
Vulnerability found	No
Reason for result	System throws exception of allowscripttagRemoting as false
Adjustment to fuzzing rules	Tried adding more injection patterns
Mitigation Steps	N/A

4. Client-side bypassing with ZAP:

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 dot 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 VCL desktop.Make sure your openMrs is running on 8082.
6. After OpenMRS starts, navigate to the OpenMRS instance at: “http://localhost:8082/openmrs-standalone “ in the Chrome browser.This should now pop up in the “Sites” list in ZAP.

Test Case ID	**Project3_Bypass_1**
Test Name	SQL Injection on Login page of OpenMRS using ZAP proxy
Steps to reproduce	1. After setting the ZAP proxy following the above steps. Open the Login page of Open MRS application in Chrome browser. URL: http://localhost:8082/openmrs-standalone/login.htm 2. Try to do a successful login attempt using admin. Username: admin, Password: “Admin123”, select Impatient ward. 3. Go to the ZAP proxy and in sites section navigate to “http://localhost:8082” -> openmrs-standalone, in this you will see a Post request for login page (Post: login.htm...). 4. Right click on it and Select breakpoint and Click OK. Breakpoint on login page will successfully set. 5. Logout from the openmrs application current session. 6. Go to ZAP Proxy and create a new Session. 7. Once the session is created successfully then go to the login page of openmrs. And Enter admin username and some password and select impatien ward. 8. Now in ZAP proxy modify the login Post request **Original request:** username=admin&password=123&sessionLocation=6&redirectUrl=%2Fopenmrs-standalone%2Freferenceapplication%2Fhome.page **Change Request:** username=admin&password=' or '1'='1&sessionLocation=6&redirectUrl=%2Fopenmrs-standalone%2Freferenceapplication%2Fhome.page 9. Once you have modified the request send this request it using Play button from the menu tab of ZAP Proxy tool. If required Submit the request again until there is no more action required in ZAP to clear the request. 10. In Chrome browser you will be redirected to the Login page of the application again and see an error message “**Invalid username/password. Please try again.**”

Description	We have modified the client request for login into OpenMRS and tried the sql injection on it. SQL Injection was failed. And suitable error message is displayed on the application.
Input Field	password
Initial User Input	“123”
Malicious Input	‘ or ‘1’=’1
Expected result	Login attempt failed with suitable error Message.
Actual Result	Login attempt failed with error message “**Invalid username/password. Please try again.**”
Test Result	Pass

Test Case ID	**Project3_Bypass_2**
Test Name	Reflected XSS on **Create/Register Patient record** in OpenMRS using ZAP proxy
Steps to reproduce	1. After setting the ZAP proxy following the above steps. Open the Login page of Open MRS application in Chrome browser. URL: http://localhost:8082/openmrs-standalone/login.htm 2. Login to Openmrs using Username: admin, Password: “Admin123”, select Impatient ward. And hit login. 3. From home page select the register patient. 4. Go to ZAP proxy and start a new Session. [If you have previous breakpoint on login page remove it from ZAP proxy app in lower pane “Breakpoint”. Deselect all the breakpoints.] 5. Go to Chrome and fill the details for a new Patient. You can select your own value, but I have selected the values: Given Name: “arpit”, “Family Name”: “kumar”, Gender: Male, BirthDate: 01 - Nov- 2000 . address: Raleigh, Phone Number: “9191234”. After filling the relative don’t Confirm and go to ZAP proxy. 6. Go to the ZAP proxy and in sites section navigate to “http://localhost:8082” -> openmrs-standalone -> registration app->registerpatien, in this you will see a Post request for Patient page (Post:submit.action..). 7. Right click on it and Select breakpoint and Click OK. Breakpoint on registration page will successfully set. 8. Logout from the openmrs application current session. 9. Go to ZAP Proxy and create a new Session. 10. Once the session is created successfully then go to the login page of openmrs. And Enter admin username and some password and select impatien ward. After successful attempt go to register patient and add some details for new patient and confirm the details. 11. Once you confirm browser hangs and ZAP will become active window.

	12. Go to ZAP proxy and go to the location mentioned in above steps 6 and modify the request as below. 13. Now in ZAP proxy modify the register patient Post request **Original request:** givenName=arpit&middleName=&familyName=kumar&preferred=true&gender=M&unknown=false&birthdateDay=01&birthdateMonth=11&birthdateYear=2000&birthdate=2000-11-01&address1=raleigh&address2=&cityVillage=&stateProvince=&country=&postalCode=&phoneNumber=91912345&relationship_type=&other_person_uuid= **Change Request:** givenName=arpit&middleName=&familyName=<script>alert("hello")</script>&preferred=true&gender=M&unknown=false&birthdateDay=01&birthdateMonth=11&birthdateYear=2000&birthdate=2000-11-01&address1=raleigh&address2=&cityVillage=&stateProvince=&country=&postalCode=&phoneNumber=91912345&relationship_type=&other_person_uuid= 14. Once you have modified the request send this request it using Play button from the menu tab of ZAP Proxy tool. If required Submit the request again until there is no more action required in ZAP to clear the request. 15. In Chrome browser you will be redirected to the register page and you see all the details. Even though the name is our script, XSS is not reflected on this page.
Description	We have modified the register patient request using ZAP proxy. We used family name as field to modify because this info is reflected after registering the page.
Input Field	Family name
Initial User Input	Family name: “kumar”
Malicious Input	<script>alert(“hello”)</script>
Expected result	XSS reflection fails and patient is registered with updated value.
Actual Result	XSS does not reflects and new patient is created and redirect to patient page.
Test Result	Pass

Test Case ID	**Project3_Bypass_3**
Test Name	Intercept the **Edit patient request and try XSS** on record in OpenMRS using ZAP proxy

Steps to reproduce	1. After setting the ZAP proxy following the above steps. Open the Login page of Open MRS application in Chrome browser. URL: http://localhost:8082/openmrs-standalone/login.htm 2. Login to Openmrs using Username: admin, Password: “Admin123”, select Impatient ward. And hit login. 3. From home page select the search patient. 4. Go to ZAP proxy and start a new Session. [If you have previous breakpoint on login page or register page remove it from ZAP proxy app in lower pane “Breakpoint”. Deselect all the breakpoints.] 5. Search for a new patient by entering any name or ID like 10000X. 6. Edit the patient details by selecting the patient and click on edit anchor. Change the details and hit Confirm. 7. Go to the ZAP proxy and in sites section navigate to “http://localhost:8082” -> openmrs-standalone -> registration app->registerpatient, in this you will see a Post request for edit Patient page (Post:editSection.page..). 7. Right click on it and Select breakpoint and Click OK. Breakpoint on registration page will successfully set. 8. Logout from the openmrs application current session. 9. Go to ZAP Proxy and create a new Session. 10. Once the session is created successfully then go to the login page of openmrs. And Enter admin username and some password and select impatien ward. After successful attempt go to seach patient and use ID 10000X and then double click on the entry and edit the details enter some details and hit Confirm. 11. Once you confirm browser hangs and ZAP will become active window. 12. Go to ZAP proxy and go to the location mentioned in above steps 7 and modify the request as below. 13. Now in ZAP proxy modify the edit patient Post request **Original request:** **givenName=rachit**&middleName=&familyName=Walker&preferred=true&gender=M&birthdateDay=11&birthdateMonth=4&birthdateYear=1971&birthdateEstimated=false&birthdate=1971-4-11 **Change Request:** **givenName=<script>alert(“hello”)</script>**&middleName=&familyName=Walker&preferred=true&gender=M&birthdateDay=11&birthdateMonth=4&birthdateYear=1971&birthdateEstimated=false&birthdate=1971-4-11 14. Once you have modified the request send this request it using Play button from the menu tab of ZAP Proxy tool. If required Submit the request again until there is no more action required in ZAP to clear the request. 15. In Chrome browser you will be redirected to the patient page and you see all the details. Even though the given name is our script, XSS is not reflected on this page.
Description	We have modified the edit patient request using ZAP proxy. We used given name as field to modify because this info is reflected after registering the page.
Input Field	Given Name
Initial User Input	Given Name: ”Rachit”

Malicious Input	<script>alert(“hello”)</script>
Expected result	XSS reflection fails and patient is edited with updated value.
Actual Result	XSS reflection fails and patient details get updated and redirect to patient page.
Test Result	Pass

Test Case ID	**Project3_Bypass_4**
Test Name	Intercept and Change Admin password in OpenMRS using ZAP proxy
Steps to reproduce	1. After setting the ZAP proxy following the above steps. Open the Login page of Open MRS application in Chrome browser. URL: http://localhost:8082/openmrs-standalone/login.htm 2. Login to Openmrs using Username: admin, Password: “Admin123”, select Impatient ward. And hit login. 3. Now go to url http://localhost:8082/openmrs-standalone/admin/users/users.list 4. Go to ZAP proxy and start a new Session. [If you have previous breakpoint on login page or register page remove it from ZAP proxy app in lower pane “Breakpoint”. Deselect all the breakpoints.] 5. Search for “ admin” by typing it in find user on name textbox and then click on Search button. 6. Now click on the admin hyperlink that appears in search results and then there type password new password in user’s password and in confirm password and then click on Save user. 7. Go to the ZAP proxy and in sites section navigate to “http://localhost:8082” -> openmrs-standalone -> admin->users, in this you will see a Post request for users form (Post:users.form..). 8. Right click on it and Select breakpoint and Click OK. Breakpoint on user password change page will be successfully set. 9. Logout from the openmrs application current session. 10. Go to ZAP Proxy and create a new Session. 11. Once the session is created successfully then go to the login page of openmrs. And then go to http://localhost:8082/openmrs-standalone/admin/users/users.list ,search for admin and then change password as mentioned in the earlier steps and click on Save user.Once you confirm browser hangs and ZAP will become active window. 12. Go to ZAP proxy and go to the location mentioned in above steps 7 and modify the request as below. 13. Now in ZAP proxy modify the users form Post request a. **Original request:** b. userId=1&person_id=1&person.names%5B0%5D.givenName=Super&person.names%5B0%5D.middleName=&person.names%5B0%

	5D.familyName=User&person.gender=M&username=&**userFormPassword=Admin1234&confirm=Admin1234**&roleStrings=Provider&roleStrings=System+Developer&secretQuestion=&secretAnswer=&property=loginAttempts&value=0&property=lockoutTimestamp&value=&action=Save+User **c. Change Request:** d. userId=1&person_id=1&person.names%5B0%5D.givenName=Super&person.names%5B0%5D.middleName=&person.names%5B0%5D.familyName=User&person.gender=M&username=&**userFormPassword=Pass1234&confirm=Pass1234**&roleStrings=Provider&roleStrings=System+Developer&secretQuestion=&secretAnswer=&property=loginAttempts&value=0&property=lockoutTimestamp&value=&action=Save+User 14. Once you have modified the request send this request it using Play button from the menu tab of ZAP Proxy tool. If required Submit the request again until there is no more action required in ZAP to clear the request. 15. In Chrome browser you will be redirected to the user management page.Now logout and login again with the tampered password provided with ZAP and you will be able to login successfully with the intercepted and changed password.
Description	We have modified the password change request using ZAP proxy. We used userFormPassword and confirm as field to modify because this info is saved as new password details for the user.
Input Field	userFormPassword ,Confirm
Initial User Input	userFormPassword : ”Admin1234” Confirm : ”Admin1234”
Malicious Input	userFormPassword : ”Pass1234” Confirm : ”Pass1234”
Expected result	Intercepted and modified Password details should not be updated to database.
Actual Result	Password details successfully intercepted,changed and updated to database for later use.
Test Result	Fail

Test Case ID	**Project3_Bypass_5**
Test Name	Intercept and Change PatientSearch queries in OpenMRS using ZAP proxy
Steps to reproduce	1. After setting the ZAP proxy following the above steps. Open the Login page

	of Open MRS application in Chrome browser. URL: http://localhost:8082/openmrs-standalone/login.htm 2. Login to Openmrs using Username: admin, Password: “Admin123”, select Impatient ward. And hit login. 3. Now go to url http://localhost:8082/openmrs-standalone/admin/patients/index.htm and there select Manage patients tab. 4. Go to ZAP proxy and start a new Session. [If you have previous breakpoint on login page or register page remove it from ZAP proxy app in lower pane “Breakpoint”. Deselect all the breakpoints.] 5. Type in “John” by typing it in find user textbox and then click on Search button. 6. Go to the ZAP proxy and in sites section navigate to “http://localhost:8082” -> openmrs-standalone -> ms>call->plaincall, in this you will see a Post request for findPatients (Post:DWRPatientService.findCountAndPatientsWithVoided.dwr) 7. Right click on it and Select breakpoint and Click OK. Breakpoint on user password change page will be successfully set. 8. Now again go to http://localhost:8082/openmrs-standalone/admin/patients/index.htm ,search for John and click enter,now browser hangs and ZAP will become active window. 9. Go to ZAP proxy and go to the location mentioned in above steps 7 and modify the request as below. 10. Now in ZAP proxy modify the findCount Post request **a. Original request:** callCount=1 page=/openmrs-standalone/admin/patients/index.htm httpSessionId= scriptSessionId=0E79A3185C8E2E3EC4DB87C0A06B1425528 c0-scriptName=DWRPatientService c0-methodName=findCountAndPatientsWithVoided c0-id=0 c0-param0=string:John c0-param1=number:0 c0-param2=number:10 c0-param3=boolean:true c0-param4=boolean:false batchId=2 **b. Change Request:** callCount=1 page=/openmrs-standalone/admin/patients/index.htm httpSessionId= scriptSessionId=0E79A3185C8E2E3EC4DB87C0A06B1425528 c0-scriptName=DWRPatientService c0-methodName=findCountAndPatientsWithVoided c0-id=0 c0-param0=string:Betty c0-param1=number:0 c0-param2=number:10

	c0-param3=boolean:true c0-param4=boolean:false batchId=2 11. Once you have modified the request send this request it using Play button from the menu tab of ZAP Proxy tool. If required Submit the request again until there is no more action required in ZAP to clear the request. 12. In Chrome browser you will be redirected to the search results pane,now you can see we have search results updated for name Betty provided with ZAP.
Description	We have modified the c0-param0=string value for getting search results for a person name.
Input field	c0-param0
Initial User Input	c0-param0=string : John
Malicious Input	c0-param0=string: Betty
Expected result	Search query should not have been intercepted,changed and reflected in search results.
Actual Result	Search query successfully intercepted,changed and reflected in search results.
Test Result	Fail

5. Security Requirements:

Requirement ID	Security Requirements
OpenMRS_SR1	Sufficient logging should be done ,with logging events written for the modules with patient information,so that it logs the details of users accessing those modules and hence it's possible to identify who accessed patients' records or created a new patient record.
OpenMRS_SR2	In case of errors or failures, limited amount of error information should be displayed to the user, which prevents leaking of any software version, exception type, stack trace.
OpenMRS_SR3	User privileges based access controls should be established so that the users with only authorized access are able to find/create patient records.
OpenMRS_SR4	Location based access controls should be established so that it can manage the access to all services based on location. For example, user who has multiple

	locations access (admin) should be able to view patients in all locations.
OpenMRS_SR5	Input fields must be properly validated on client and server side to prevent SQL Injection and Cross Site Scripting attacks. We should also constrain input, sanitize input, reject known bad input for the fields using for searching patient's records, and creating new patient record.
OpenMRS_SR6	Every request made to find/create patient records must use HTTPS protocol through SSL/TLS certificates which allows an encrypted secure connection between client and server hence safeguarding from man-in-the-middle attack.
OpenMRS_SR7	Strong access control measures like role based applications module access,limited number of people with admin privileges, should be implemented so that patients records must be kept confidential and not disclosed to an unauthorized party which protects from sensitive data exposure vulnerability/attack.
OpenMRS_SR8	Users passwords should be protected by salted password hashing while registering user or in login module, safeguarding from dictionary and brute force attacks, lookup tables, reverse lookup tables, rainbow tables. This is for the login/register module.
OpenMRS_SR9	Data protection is required which means platform security configuration, use of encryption at rest and in transit and entire life cycle secure is needed. This requirement is for the entire system.
OpenMRS_SR10	Audit policies should be detailed enough to be useful, with specificities defined for each of the modules functionalities. This is for the entire system.
OpenMRS_SR11	Data consistency and correctness of health care information in the database are key requirements. Incorrect data can have bad consequences, such as incorrectly diagnosing a patient. This is for the database system.
OpenMRS_SR12	Privacy of protected health information should be implemented as per HIPAA Privacy Rule which includes individual's right to adequate notice of uses and disclosures of his or her protected health information that may be made by the covered entity and the individual's rights and the covered entity's duties with respect to the protected health information. This requirement is for the entire system.

References: https://www.himss.org/library/interoperability-standards/security-standards

Project Part 4:

1. Architectural Design Principles

i) The following test case violates the principle of "Separation of Privilege" which states that:

"Software should not grant access to a resource, or take a security-relevant action, based on a single condition."

Test Case ID	Test_1
Test Name	**Separation of super user privilege to multiple users**
Steps for test case	i)Go to Admin Home Page using Admin credential. (URL: http://152.46.16.55:8082/openmrs-standalone/referenceapplication/home.page) : Username: admin, PW: Admin123 ii) Click on System Administration on the home page and then click on "Manage Accounts". iii) On the next page Select "Add new Account". Add a user Family Name: "Jain" Given Name: "Anuj",Male. And Select Add User account checkbox. Enter Username "januj" and password as "Ncsu1234", privilege as "full". Uncheck force password change. In capability only select "Administer System" and save the detail. And new user is created. iv) Logout from this user and login with new user credential. UN: "januj" and PW: "Ncsu1234" InPatient Ward. v) On Home Page select "System Administrator" and then Manage Account module on next page. vi) Click on "Add New Account". Add a user Family Name: "Jain" Given Name: "Sanskar",Male. And Select Add User account checkbox. Enter Username "sjain" and password as "Ncsu1234", privilege as "full". Uncheck force password change. In capability only select "Has Super User Privileges" and save the details. And new user is created. vii) Logout from this user account. viii) Login to new user account with new sjain's credential. You are able to successfully able to create a new super user.
Expected Result	New User "sjain" should have not given the Super user privilege at the moment. There should be a pipeline mechanism until all user in that pipeline don't approve user should not be given very elevated privilege like "Super user"

Test Case Status	Failed
Mitigation to be taken by OpenMRS	Various users of the application should be authorized to approve new user request such as super user. Until all of them don't Approve the request the new user creation request will be in Pending state. Once they approve the request new user will be Created. This is how we can separate the sensitive operation to Take place with multi user intervention.

ii) The following test case violates the principle of "Complete Mediation" that states:
Software should validate every access to object to ensure that access is allowed

Test Case ID	Test_2
Test Name	**Elevation of privilege of standard user**
Steps for test case	i)Go to Admin Home Page using Admin credential. (URL:http://152.46.16.55:8082/openmrs-standalone/login.htm) ii) Click on System Administration on the home page and then click on"Advanced Administration". iii) On the next page Select "Manage Users" Anchor link. iv) From the Role drop-down menu select "Organizational: Nurse" as roles. It will list Nurse user Select the hyperlink under System-ID 4-2.that brings to next page. v) Change the username and password from the input item User password's and confirm it in the next password box the same Password. I have selected the password "Ncsu1234". And select the save user button to save the information. vi) Now login as nurse,where on the dashboard you don't have access to System administration option. vii) Now open manage users link which is available to admin user, http://152.46.16.55:8082/openmrs-standalone/admin/users/users.list?name=admin&role=&action=Search. viii) Now type admin in find user or name field, in search result admin records will appear, click on admin link appearing in first column of the record of the search result. viiii) Now in the page that appear, we can change the admin's password and other details,so let's change the password from Admin123 to Admin1234 and save it ix) Now logout and login with username as admin and

	password as Admin123, it will fail and now login with password Admin1234. x) So here , a standard user with not having access to manage the users did changed the admin account details.
Expected Result	User 'nurse' not having access to admin pages of manage users, shouldn't have been able to access it and ability to also modify the data over there
Test Case Status	Failed
Mitigation to be taken by OpenMRS	Various level of users of the application should be authorized to access different parts of the application after authentication.

iii) The following test case violates the principle of "Fail Safe defaults" that states:
The initial state should be to deny access unless access is explicitly required. Then, unless software is given explicit access to an object, it should be denied access to that object and the protection state of the system should remain unchanged.

By default, weak passwords should not be allowed

Test Case ID	**Test_3**
Test Name	**Weak password acceptance in Register Patient**
Steps for test case	i) Go to http://152.46.16.55:8082/openmrs-standalone/index.htm, there i was logged in as admin ,then click on admin dropdown top bar of that page,then Myaccount button will appear,so click on it. ii) Next click on Change password button. iii) Now in the page that appears, provide the old password, and provide a weak new password such as **'Test1234'**, and then save it. iv) Now logout and login again with new password
Expected Result	The application should not have allowed to set a weak password while changing the password, as it makes system vulnerable to dictionary and brute-force attacks.
Test Case Status	Failed
Mitigation to be taken by OpenMRS team	The password rules for authentication should be strong and not easily open to dictionary and brute-force attacks. A combination of upper and lower case characters with a mix of alpha-numeric characters that don't usually form a sequence is a good password rule.

2. Usable Security Principles:

2.1 Appropriate Boundaries:

OpenMRS violates Appropriate boundaries principle for User Interaction Design in Secure Systems. As per the principle, the interface should distinguish objects and actions along boundaries that relate to important issues such as “need to know” or “least privilege”. In OpenMRS doctors can register a patient given that he is aware of the register patient request link.

Test Case ID	USP1
Description	Low privilege boundary between doctor and admin.
Steps:	1) Login to OpenMRS using admin credentials and navigate to “Register Patient” section and copy the url from browser which is http://localhost:8081/openmrs-standalone/registrationapp/registerPatient.page?appId=referenceapplication.registrationapp.registerPatient 2) Now log out from admin and login using doctor credentials. 3) You will find only three options on the home menu viz “Find Patient Record”, “Active Visits”, “Appointment Scheduling”. 4) Paste the link copied from “Register Patient” section and enter. 5) You’ll be redirected to “Register Patient” section even though you’re logged in as doctor.
Expected Results	Doctor should not able to access “Register Patient” section/module.
Actual Results	Doctor is able to access “Register Patient” section/module.
Result	FAIL

2.2 Explicit Authorization:

OpenMRS violates Explicit Authorization principle for User Interaction Design in Secure Systems. Explicit Authorization means the 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. As mentioned in the above test scenario the doctor is able to access the admin console by just manually inserting the link in the browser. Then the doctor is able to create a new user and provide full privileges. During this process the user should have been authorized while providing full privileges to the newly created user.

Test Case ID	USP2
Description	Doctor able to create user by accessing admin console.
Steps:	1) Login to OpenMRS using Doctor credentials and visit the below link by entering it in browser search. (http://localhost:8081/openmrs-standalone/admin/users/users.list) 2) Click on Add user and create a new user by entering user information and selecting Roles as “Privilege Level Full” and save. 3) Log out from doctor’s account and login as the created user. 4) Try login using newly created user’s credentials.
Expected Results	Doctor shouldn’t have the access to create new users and the newly created user shouldn’t be allowed to login.
Actual Results	The newly created user was able to login.
Result	FAIL

2.3 Revocability + Expected Ability:

OpenMRS violates Revocability and Expected Ability principle for User Interaction Design in Secure Systems. Revocability principle states the admin interface should allow the user to easily revoke authorities that the user has granted and Expected Ability principle states the interface must not generate the impression that it is possible to do something that cannot actually be done. If a user is already logged in the system and admin downgraded or changed the user privileges/roles, it doesn’t reflect in the current session and the user is under wrong impression that it’s still possible to perform activities.

Test Case ID	USP3
Description	Doctor able to perform activities in current session even after privileges being revoked.
Steps:	1) Login to OpenMRS via browser 1 say Firefox using system admin credentials and login to OpenMRS via browser 2 say Google Chrome using nurse credentials.

	2) In browser 1 (for admin) navigate to System Administration → Advanced Administration → Manage Users → Search for nurse “Jane Smith” and change nurse’s role from Organizational: Nurse to Organizational: Doctor. 3) In browser 2 (for nurse) refresh the page, the nurse is able to navigate to “Capture Vitals” module and search.
Expected Results	Nurse user should not be able to view/access “Capture Vitals” module after the her role has been changed.
Actual Results	Nurse user is able to view/access “Capture Vitals” module.
Result	FAIL

3. Protection Poker

FUNCTIONAL REQUIREMENTS:

1) **Addition of Payment records with Patient’s Past Visits :** Patient payment records can be maintained as a separate task managed by admin and integrated with the patient past visits
2) **Maintaining Patient immunization history :** Patient immunization history could be maintained along with other patient details
3) **Saving Patient Insurance records :** Patient health insurance records should be saved in the database and should be used along with the appointments functionality as the same insurance is likely to be used for a long period of time.
4) **Addition of emergency ward location :** Emergency ward could be integrated as another location in the OpenMRS database
5) **Maintaining Patient referrals records :** New patient referrals records could be maintained

Table 1 : Database tables value points

Database table	Value points	Usage in Requirement No.
Visit	Sachin: 13 Abhash: 13 Arjun: 8 Aishwarya: 13 **Team:** 13	1,2,3,5
Patient	Sachin: 20	1,3,5

	Abhash: 8 Arjun: 8 Aishwarya: **Team:** 20	
Visit_type	Sachin: 8 Abhash: 8 Arjun: 8 Aishwarya: 5 **Team:** 8	2
Visit_attribute	Sachin: 8 Abhash: 8 Arjun: 8 Aishwarya: 8 **Team:** 8	2
Visit_attribute_type	Sachin: 8 Abhash: 5 Arjun: 8 Aishwarya: 5 **Team:** 8	2
form	Sachin: 13 Abhash: 13 Arjun: 8 Aishwarya: 13 **Team:** 13	2,3
form_field	Sachin: 13 Abhash: 13 Arjun: 8 Aishwarya: 8 **Team:** 13	2,3
field_answer	Sachin: 13 Abhash: 13 Arjun: 13 Aishwarya: 13 **Team:** 13	2,3
location	Sachin: 5 Abhash: 5 Arjun: 5 Aishwarya: 5 **Team:** 5	4
location_tag	Sachin: 5 Abhash: 5 Arjun: 3 Aishwarya: 3	4

	Team: 5	
location_attribute	Sachin: 5 Abhash: 5 Arjun: 3 Aishwarya: 5 **Team:** 5	4
Patient_identifier	Sachin: 13 Abhash: 13 Arjun: 8 Aishwarya: 5 **Team:** 13	5

Table 2: Database tables used by requirements

Requirement	**Table used**	**Value points of Table**	**Max value point**
1	Visit Patient	13 20	20
2	Visit Visit_type Visit_attribute Visit_attribute_type form form_field field_answer	13 8 8 8 13 13 13	13
3	Visit Patient form form_field field_answer	13 20 13 13 13	20
4	location location_tag location_attribute	5 5 5	5
5	Visit Patient Patient_identifier	13 20 13	20

Table 3 : Security risk
(Security risk = Ease points x Value points)

Requirement	Ease of attack points	Value points	Security risk	Rank
1	Sachin: 8 Abhash: 8 Arjun: 8 Aishwarya: 5 **Team:** 8	20	160	2
2	Sachin: 3 Abhash: 3 Arjun: 3 Aishwarya: 3 **Team:** 3	13	39	5
3	Sachin: 5 Abhash: 5 Arjun: 8 Aishwarya: 8 **Team:** 5	20	100	3
4	Sachin: 13 Abhash: 13 Arjun: 8 Aishwarya: 13 **Team:** 13	5	65	4
5	Sachin: 13 Abhash: 13 Arjun: 13 Aishwarya: 8 **Team:** 13	20	260	1

Evidence and commentary on rankings:

With protection poker, in order to assign some value points we did found out database tables for corresponding requirements , since the number of tables and their strategic importance will in turn determine their value and ease points, and as a corollary will impact overall security risk and rankings.

Also for some of the features there had been many tables with interdependent relationships which does make complexity of those relationships a major factor to assign value or ease points.

Now since security risk was dependent on value points and ease points, which in turn depends on number of tables and their associated data composition and relational complexity, it also determined the complexity of feature.
Feature 5 is more complex relative to rest of the features, due to its higher value points attributing to important tables being used for it and also relatively higher ease points with those tables being relatively easy to target.
Feature 1 is relatively simpler as compared to feature 5 attributing to lesser ease points with tables comparatively harder to access a compared to feature 5.Similarly, requirement 3 although having same value points does have lesser ease points thus less security risk as compared to requirement 5, because of the tables used in the requirement are harder to access because of its complex interrelationships.Moreover for requirement 4, although there is higher ease points, there is less value to these records comparatively, which overall makes its security risk lower than requirement 3. Lastly, requirement 2 has relatively lower ease points and value points than requirement 4, which did make it least risky requirement.

META Plan :

Requirement No.	Applicable META	Remarks
1.	M	Payment records are extremely sensitive records and its disclosure can result in malicious usage resulting in monetary loss to the patients and also hospitals if patient sues them for the monetary compensation for it. This risk can be mitigated with stronger encryption and hashing of sensitive payment data stored in database.Also these records , while presented for view on UI can be hidden to a extent to not reveal all the PII(Personally Identifiable Information).
2.	M	Immunization records can be encrypted in the database. Also ,strong access control lists can be maintained for roles based access to the tables containing immunization records data. Moreover, for the encryption algorithm, we will use a standard and secure algorithm like RSA or AES-256 , instead of reinventing the wheel with our own encryption algorithm and in turn compromising the system security.
3.	M	Patient insurance records are very sensitive piece of information, and its revelation will result in non-compliance of HIPAA regulations ,which in turn compromises the organisation’s integrity and patient

		details. To mitigate this strong encryption algorithm and Access Control Lists can be maintained to limit the access of insurance information availability to the number of application roles and users, thus limiting the risk of disclosure and security failure.
4.	M	Locations impact patients visit data indirectly, as if patient visits are incorrectly linked to a non-desired location, then it will impact the integrity and consistency of patient medical records. This risk can be mitigated by limiting this privilege to admin access user, so that limited number of users have access to it and thus minimise the risk involved in it.
5.	M	Patient referral records are important in context of patient medical history ,often adhering to HIPAA regulations,its disclosure can result in invasion of privacy. These records can be made more secure with sensitive data encrypted in database and limiting the access to these records to a specific set of users,thus limiting the risk with enforcement of stronger access control lists.

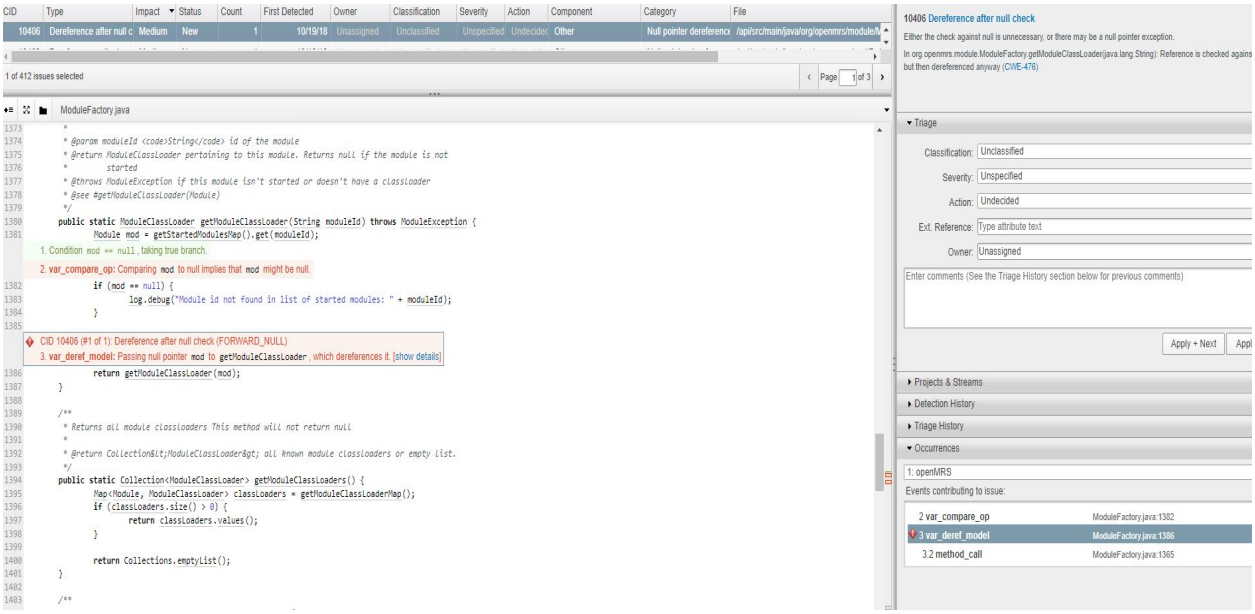
4. Bug Fixes:

For Project Part 4, here are the bug fixes we had developed:

i) ID : Bug_fix_1

Test Case : Dereference after null check (Check against null is unnecessary or there may be NULL pointer exception)

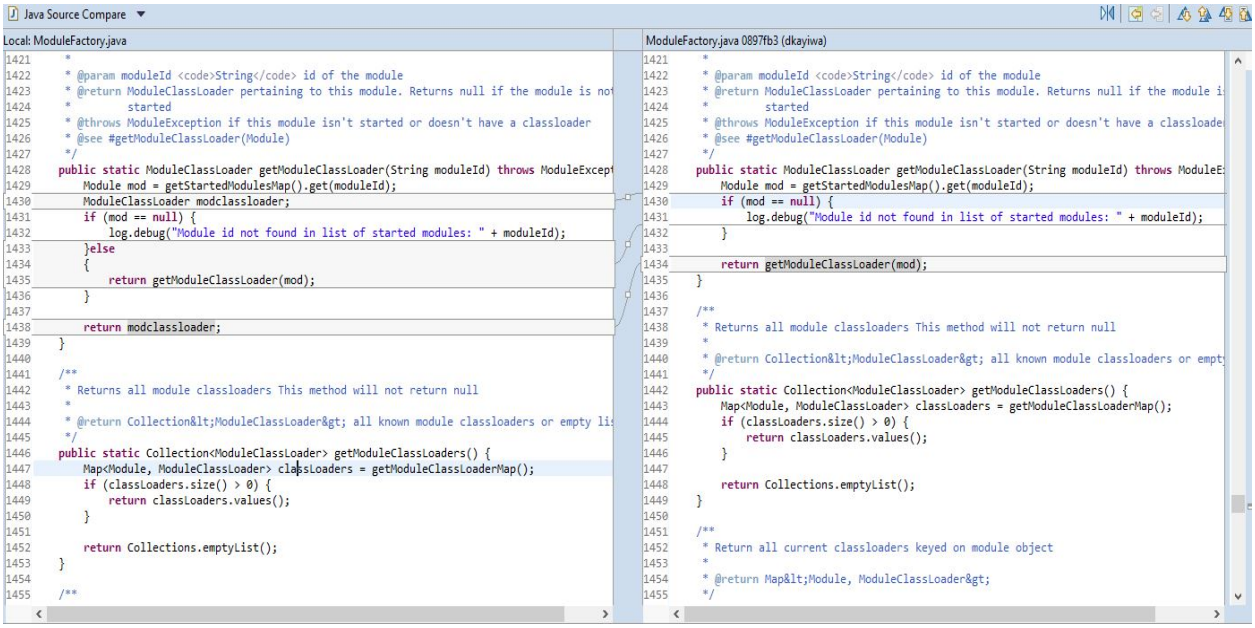
Reference : Test case 8 , in Report 3
CID: 10406 in Coverity report



Tool used : Coverity

Diff of code :

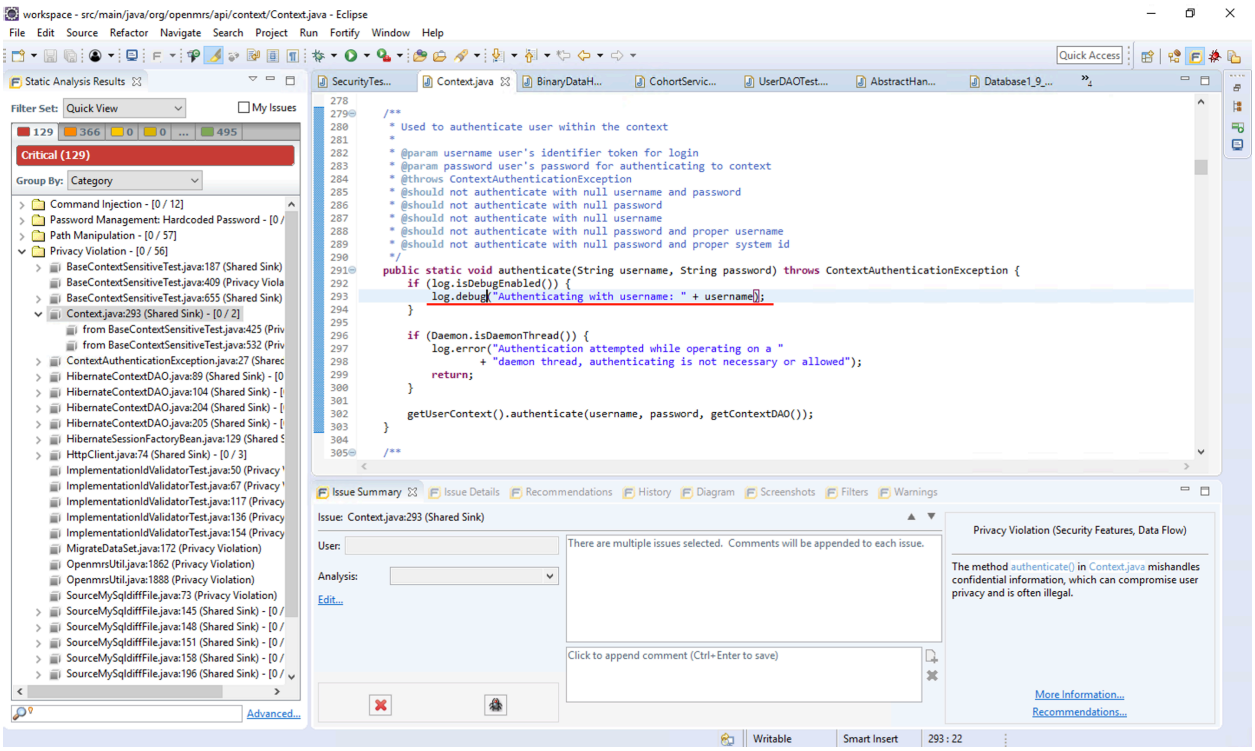
Fixed code: **Buggy Code:**



ii) ID : Bug_fix_2

Test Case : Privacy violation: (This method mishandles the confidential information which can compromise the user privacy)

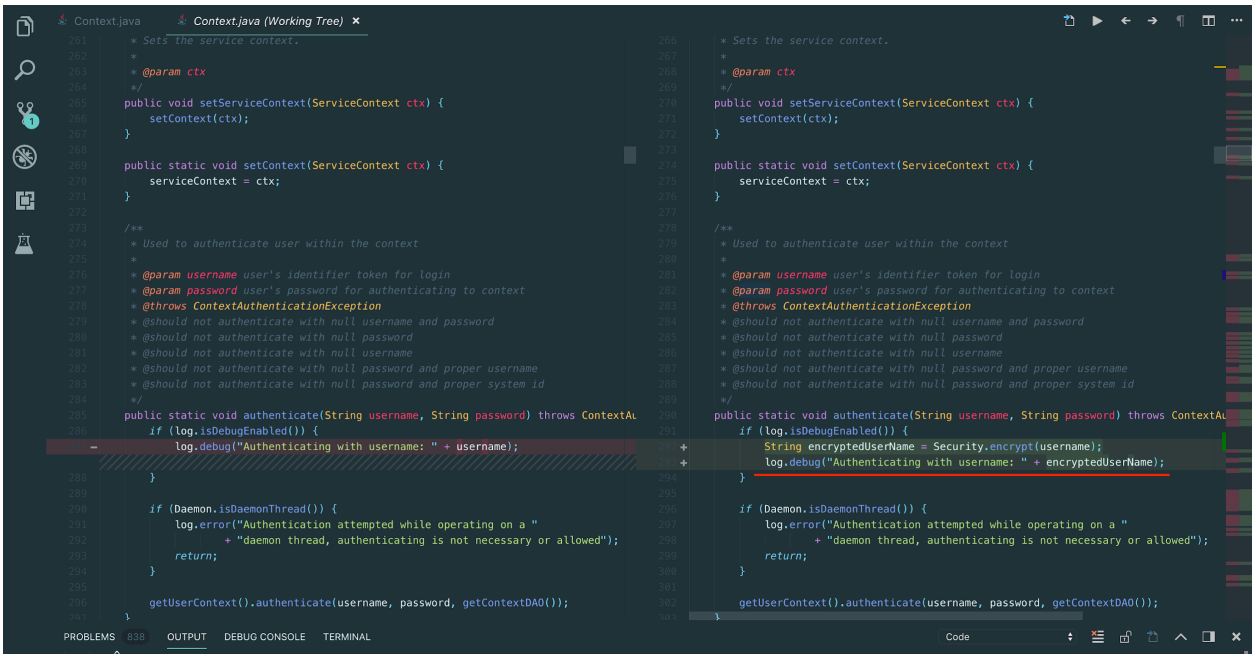
Reference : Privacy Violation in Context.java as found in Fortify report



Tool used : Fortify

Diff of code :

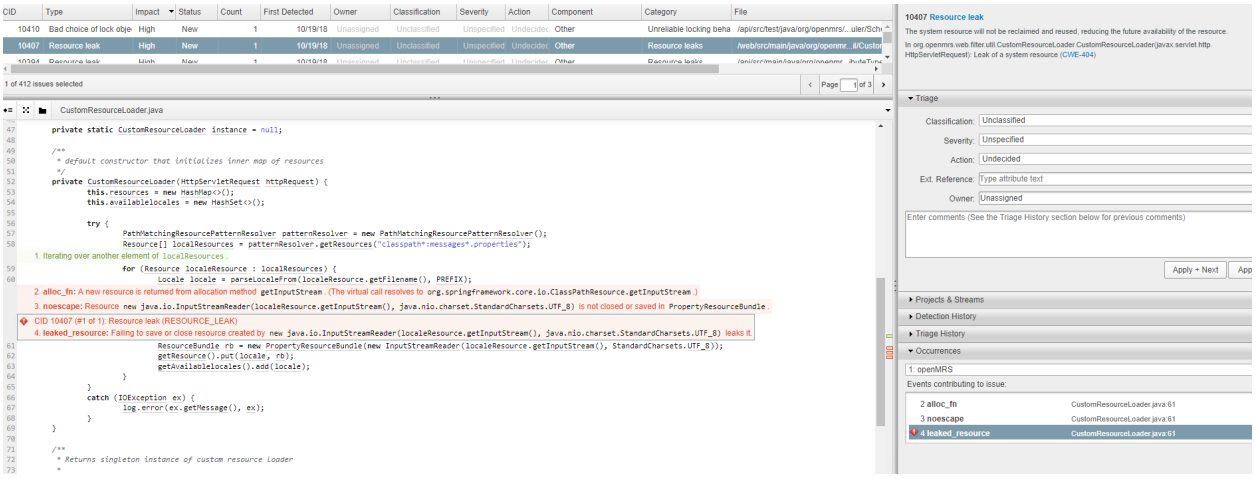
Buggy Code: **Fixed code:**



iii) ID : Bug_fix_3

Test Case : Resource leak (System resource is not reclaimed and reused reducing the future availability of the resource)

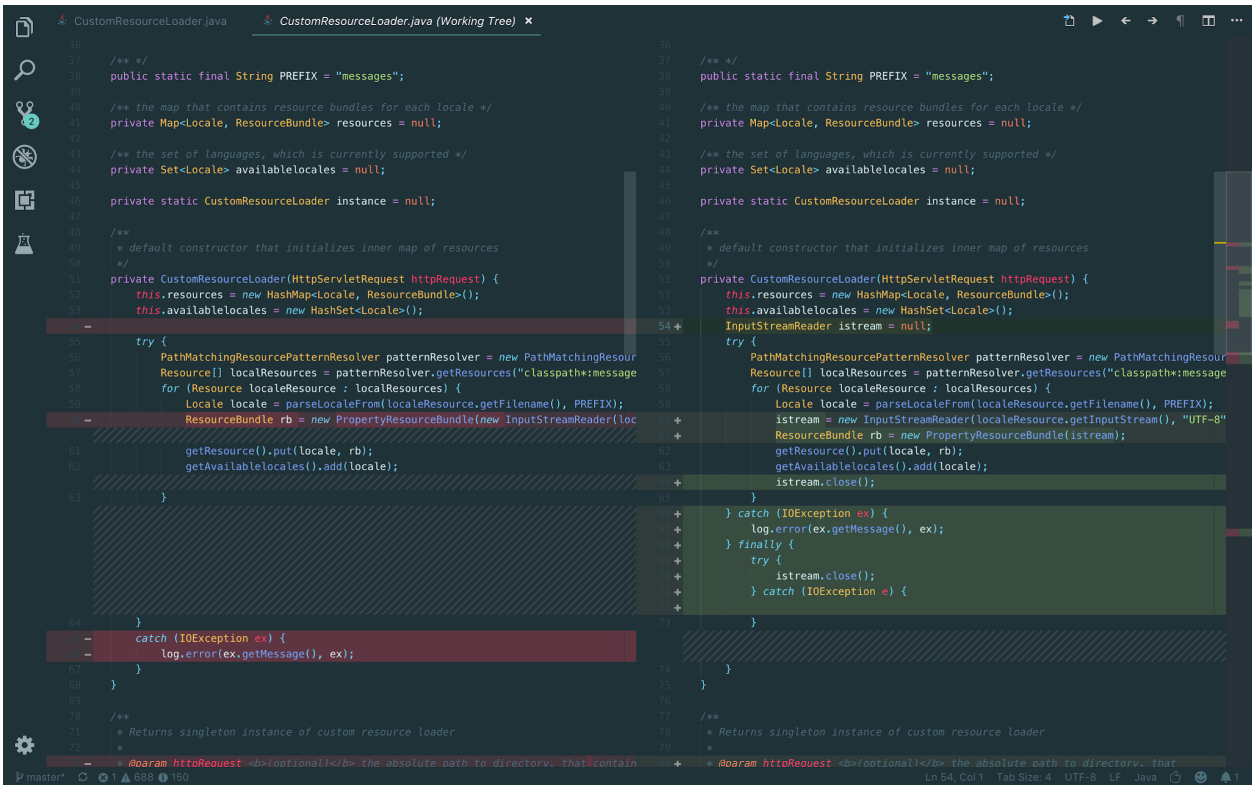
Reference : Test case 2 , in Coverity Bugs sections of Report 3
CID: 10407 in Coverity report



Tool used : Coverity

Diff of code :

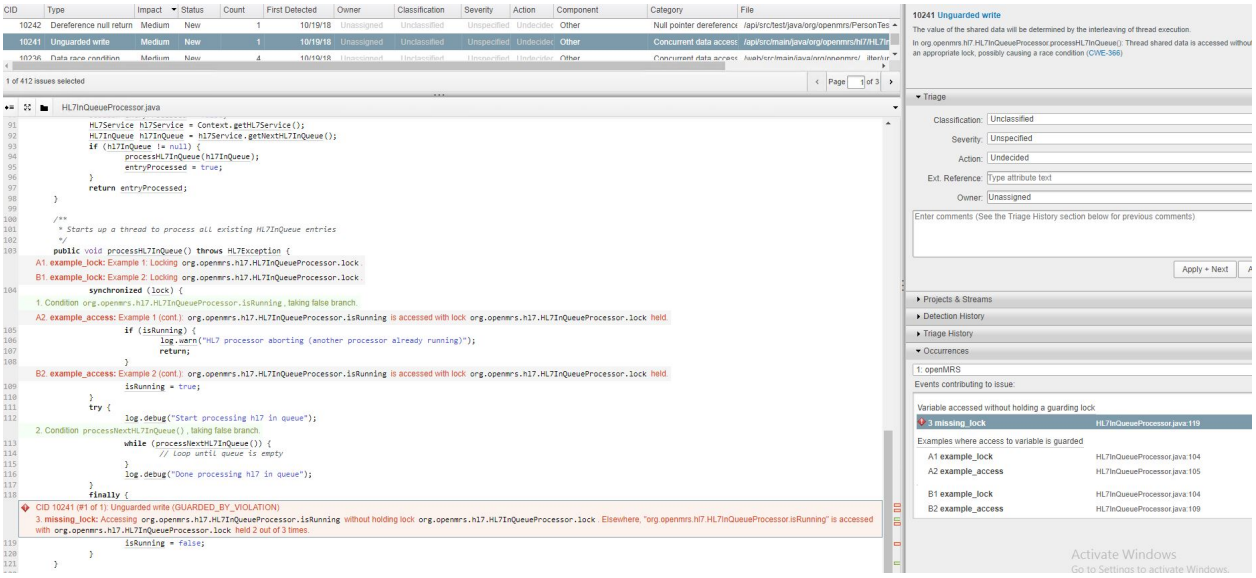
Buggy Code: **Fixed code:**



iv) ID : Bug_fix_4

Test Case : Unguarded write (Value of the shared data is determined by the interleaving of the thread execution, which can cause a Race condition which might lead to a deadlock state.)

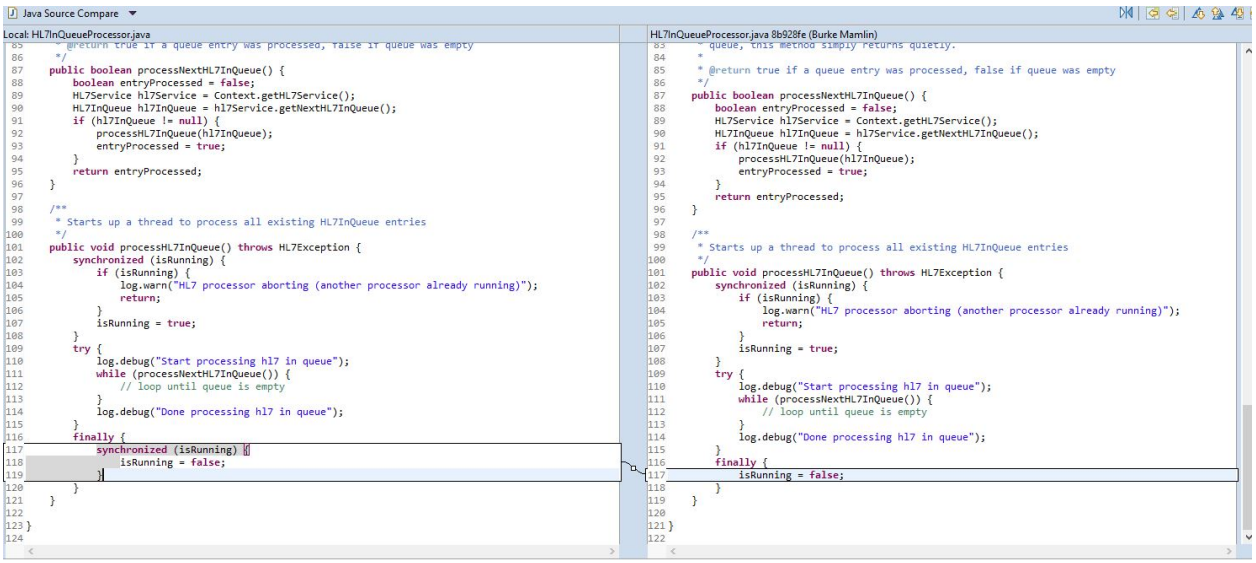
Reference : CID: 10241 in Coverity report



Tool used : Coverity

Diff of code :

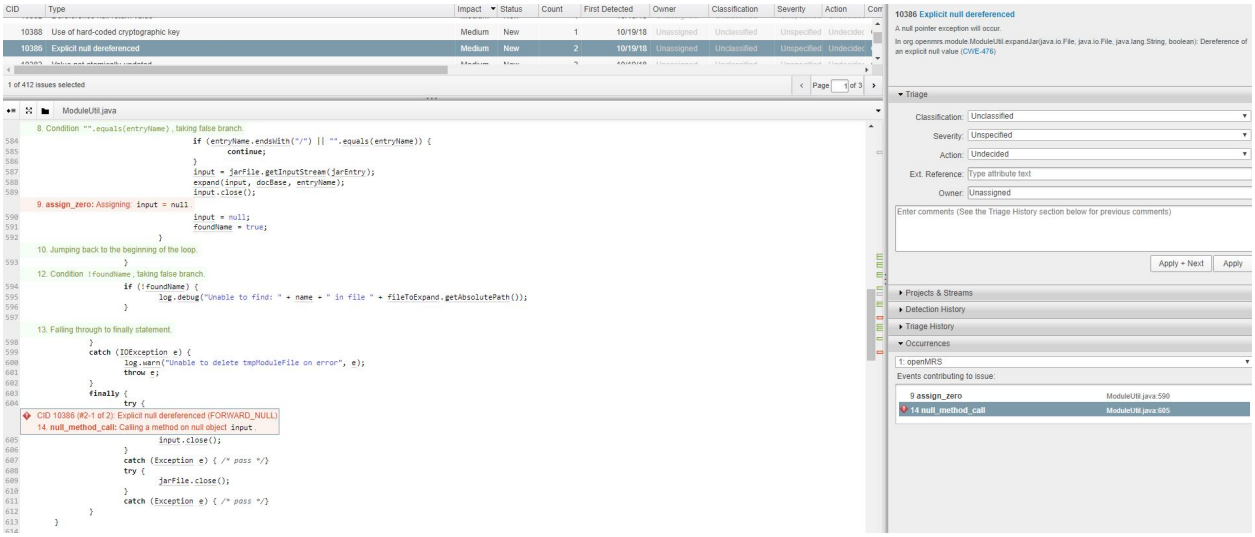
Fixed code: **Buggy Code:**



v) ID : Bug_fix_5

Test Case : Explicit null dereferenced (A null pointer exception will occur due to rerefereing of an explicit null value)

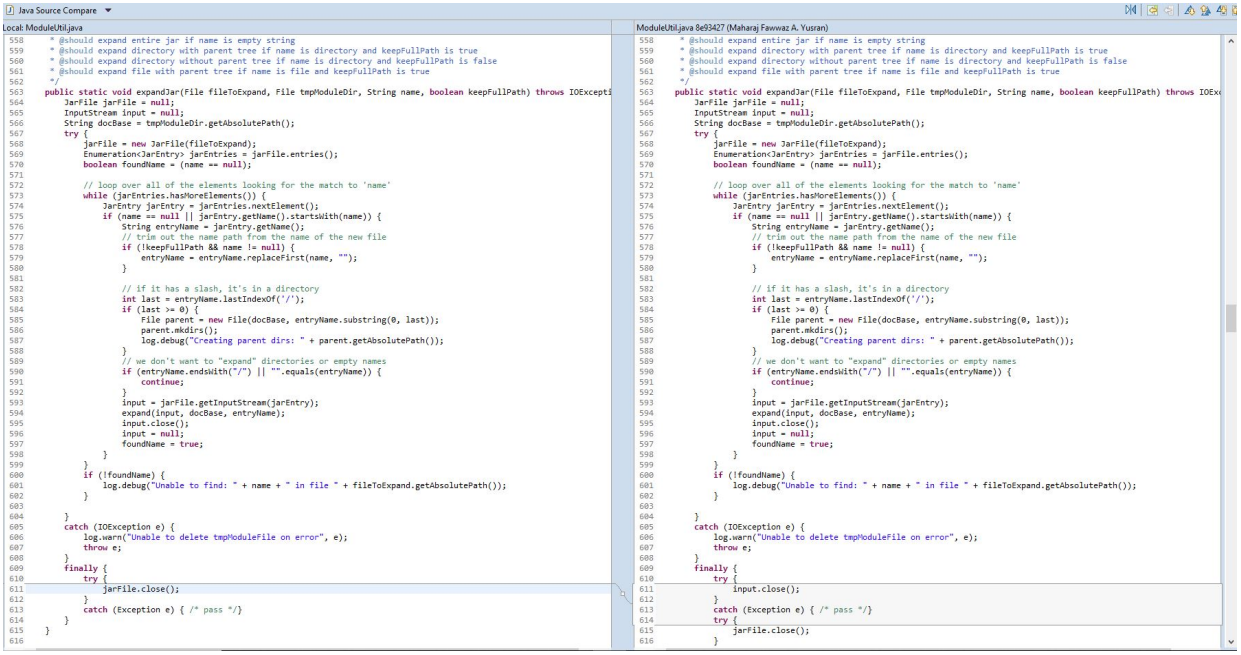
Reference : CID: 10386 in Coverity report



Tool used : Coverity

Diff of code :

Fixed code: **Buggy Code:**

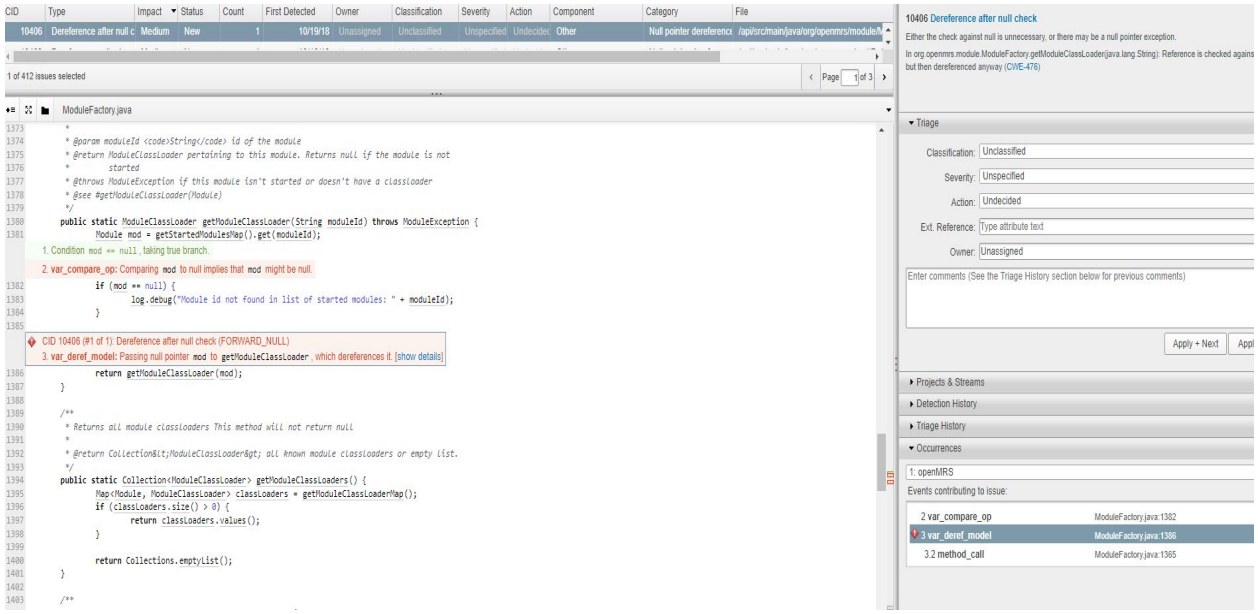


3)

i) Vulnerability 1 :

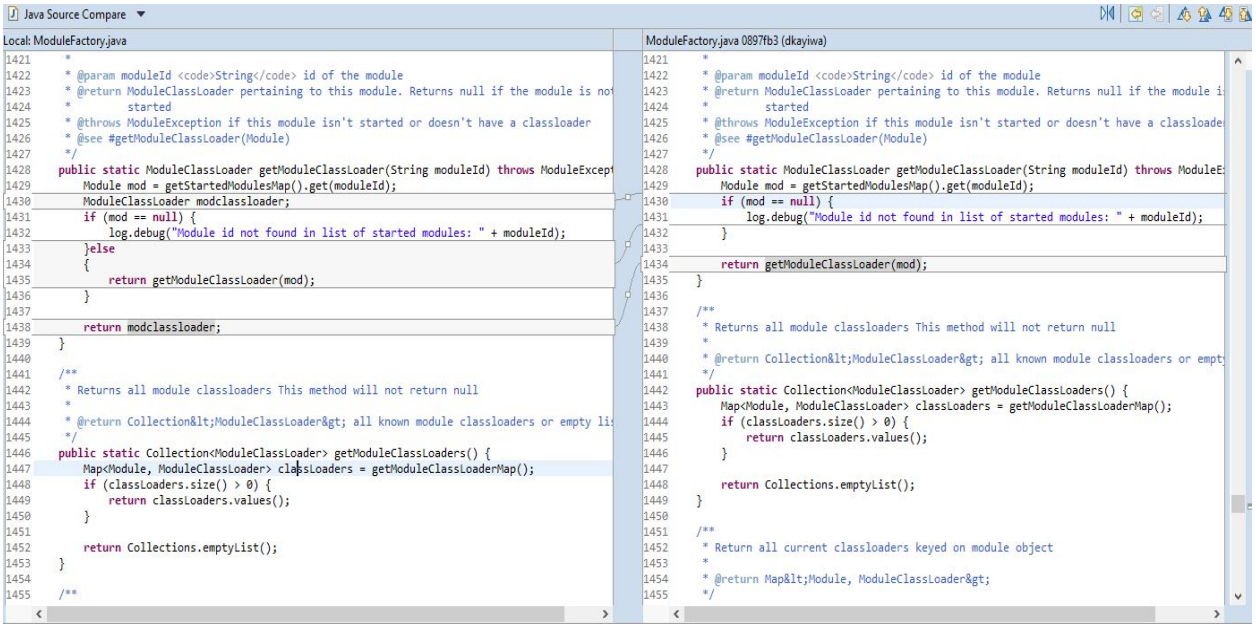
ID : Bug_fix_1

Description : Dereference after null check (Check against null is unnecessary or there may be NULL pointer exception) as observed in ModuleFactory.java's method getModuleClassLoader referenced as CID: 10406 in Coverity report.
Here reference is checked against null but then dereferenced anyway.



Diff of code :

Fixed code: **Buggy Code:**



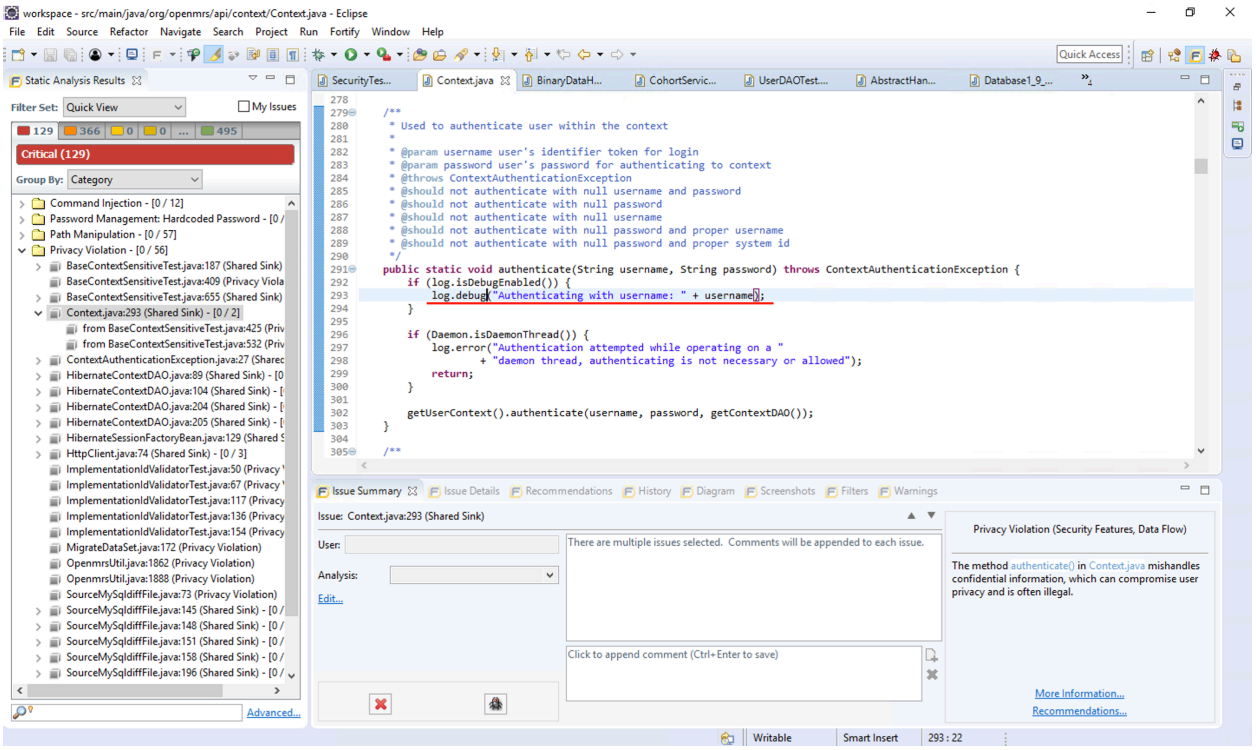
Suggested Mitigation to OpenMRS team:

getModuleClassloader should be returned only when mod is not equal to null, so return statement for getmoduleclassloader should be added as else statement of the if statement of checking mod equals to null.
Also thereby mod should be returned from the method as it will be showing error once we moved the return statement of getmoduleclassloader in the else statement.

ii) Vulnerability 2:

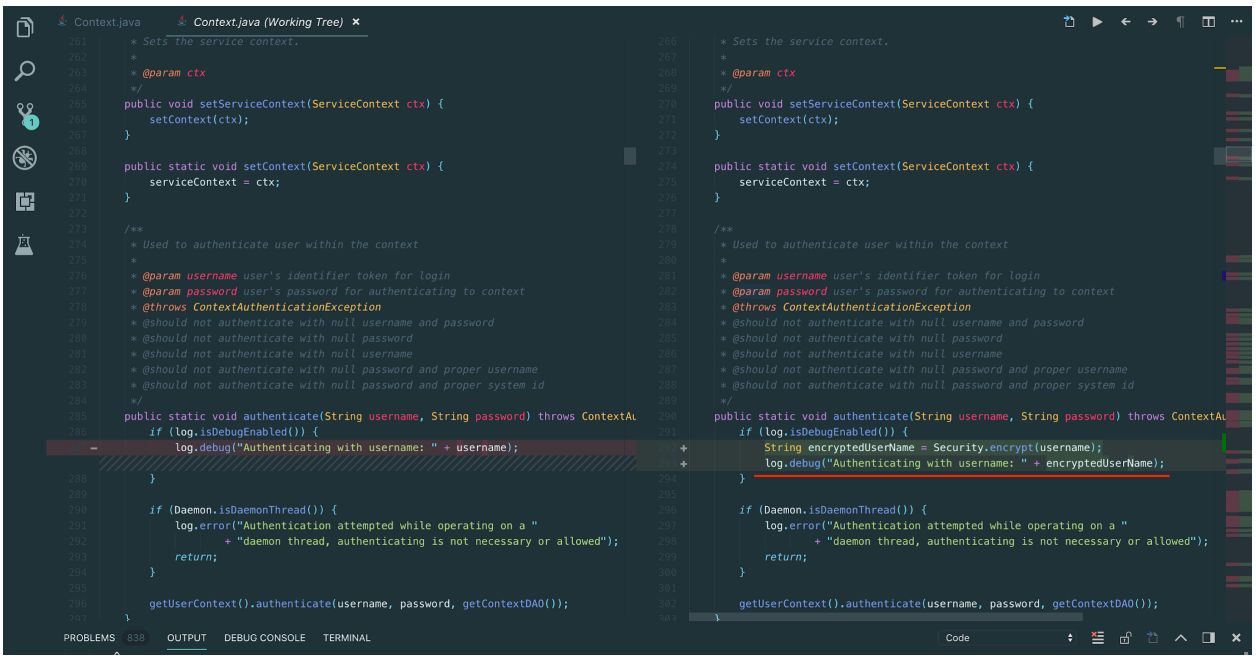
ID : Bug_fix_2

Test Case : Privacy violation: (This method mishandles the confidential information which can compromise the user privacy).
Description: As referenced from Privacy Violation in Context.java from fortify report we found that in authenticate method of context.java file there is a log where username is getting printed in plain text format. As this is categorized into sensitive information and during the attack can lead to exposed the username of the people in the OpenMRS system.



Buggy Code:

Fixed code:



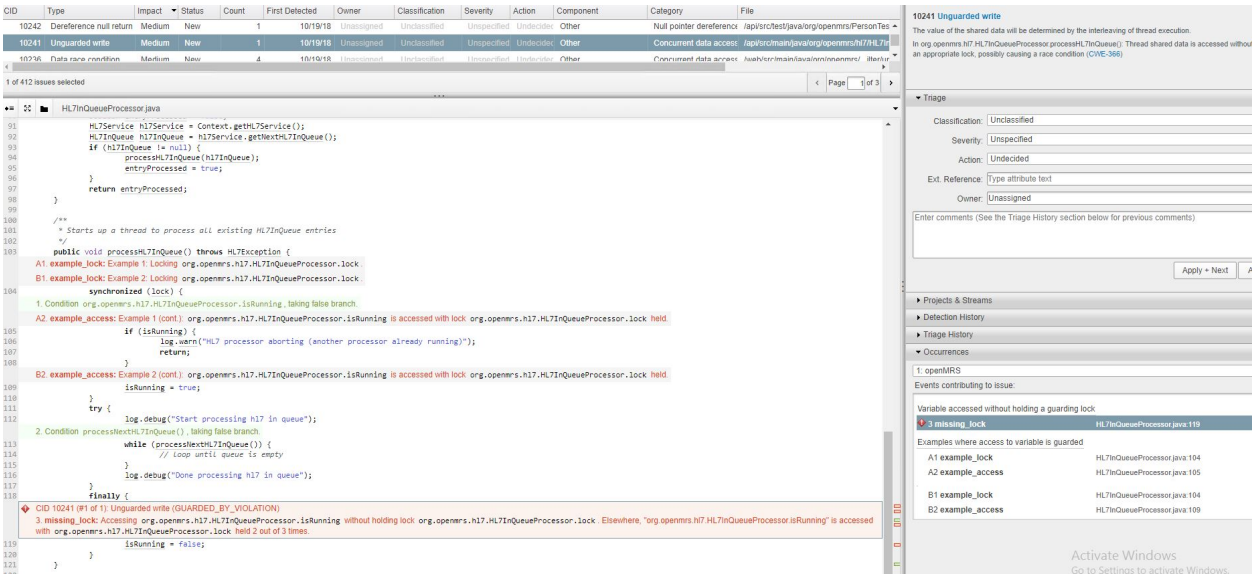
Suggested Mitigation to OpenMRS team:

As username is a very sensitive information, we can encrypt this username before printing into the log file. To encrypt this we have used Security.encrypt method from Java library. Suggested fix is shown in the diff above.

iii) Vulnerability 3 :

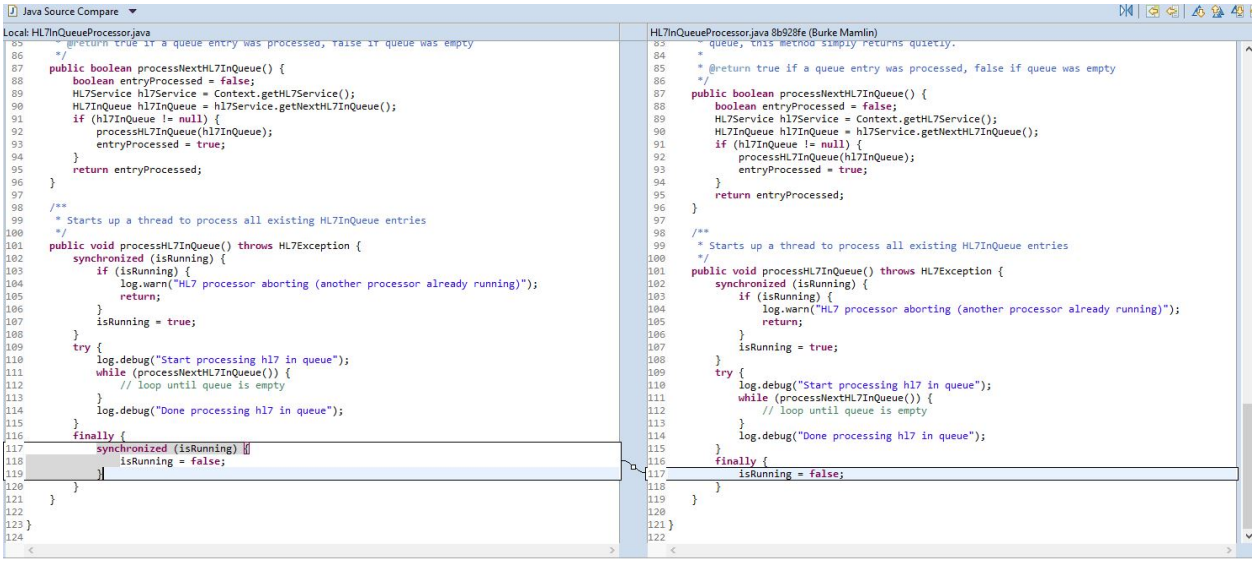
ID : Bug_fix_3

Description : Unguarded write (Value of the shared data is determined by the interleaving of the thread execution, which can cause a Race condition which might lead to a deadlock state.) This issue is present in method processHL7Inqueue of HL7InQueueProcessor.java in [Referenced in CID: 10241 of Coverity report]



Diff of code :

Fixed code: **Buggy Code:**



Suggested Mitigation to OpenMRS team:

Synchronise isRunning variable to prevent race condition, as previously thread shared state is accessed without an appropriate lock. So synchronising the isRunning variable, will prevent the race condition and hence the deadlock condition.

2. Bug Report:

Vulnerability #1:

1) Vulnerability Name - Key Management (Hardcoded Encryption Key)

2) Business Impact - The use of a hard-coded cryptographic key significantly increases the possibility that encrypted data may be recovered. Once the code is in production, the encryption key cannot be changed without patching the software. If the account that is protected by the encryption key is compromised, the owners of the system will be forced to choose between security and availability.

3) Affected Component - The encrypt functionality in /util package and file Security.java.

4) Description - The vulnerability was found in util/OpenmrsConstants.java where the constant ENCRYPTION_KEY_SPEC is defined and used in Security.java during encryption.

5) Result - If hard-coded cryptographic keys are used, it is almost certain that malicious users will gain access through the account in question.

6) Mitigation - In design Phase, usage of prevention schemes mirror that of hard-coded password storage.Also a properties file can be maintained to store the cryptographic key and that properties file should be access restricted with access to a limited group of people.

```
/**
 * Encryption properties; both vector and key are required to utilize a two-way encryption
 */
public static final String ENCRYPTION_CIPHER_CONFIGURATION = "AES/CBC/PKCS5Padding";

public static final String ENCRYPTION_KEY_SPEC = "AES";

public static final String ENCRYPTION_VECTOR_RUNTIME_PROPERTY = "encryption.vector";

public static final String ENCRYPTION_VECTOR_DEFAULT = "9wyBUNglFCRVSUhMfsTa3Q==";

public static final String ENCRYPTION_KEY_RUNTIME_PROPERTY = "encryption.key";

public static final String ENCRYPTION_KEY_DEFAULT = "dTfyELRrAICGDwzjHDjuhw==";
```

Vulnerability #2:

1) Vulnerability Name - Weak SecurityManager Check: Overridable Method

2) Business Impact - Attackers can override security checks and can impact both the business and its consumers harming both in monetary and reputation terms.

3) Affected Component - The vulnerability was found in util/OpenmrsSecurityManager class where getCallerClass() method is defined public.

4) Description - Non-final methods that perform security checks may be overridden in ways that bypass security checks.

5) Result - Make sure any methods that perform security operations (e.g. methods from SecurityManager or AccessController) are declared in final classes or the methods themselves are declared final. Exposing critical functionality essentially provides an attacker with the privilege level of the exposed functionality. This could result in the modification or exposure of sensitive data or possibly even execution of arbitrary code.

6) Mitigation - If you must expose a method, make sure to perform input validation on all arguments, limit access to authorized parties, and protect against all possible vulnerabilities. Identify all exposed functionality. Explicitly list all functionality that must be exposed to some user or set of users. Identify which functionality may be accessible to all users, restricted to a small set of privileged users, and prevented from being directly accessible at all

```
32      public Class<?> getCallerClass(int callStackDepth) {
33          if (callStackDepth < 0) {
34              throw new APIException("call.stack.depth.error", (Object[]) null);
35          }
36
37          //SecurityManager may appear more than once in classContext
```

Vulnerability #3:

1) Vulnerability Name - Server-Side Request Forgery

2) Business Impact - It can make attackers steal the data and also making the system unavailable to actual users costing business both money and reputation.

3) Affected Component - The function openConnection() on line 714 initiates a network connection to a third party system using user-controlled data for resource URI. An attacker may leverage this vulnerability to send a request on behalf of the application server since the request will originate from the application server's internal IP address.

4) Description - A Server-Side Request Forgery occurs when an attacker may influence a network connection made by the application server. The network connection will originate from the application server's internal IP and an attacker will be able to use this connection to bypass network controls and scan or attack internal resources that are not otherwise exposed.

5) Result - An attacker will be able to leverage this hijacked network connection to perform the following attacks:- Port Scanning of intranet resources, bypass firewalls, attack vulnerable programs running on the application server or on the intranet, attack internal/external web applications using Injection attacks or CSRF.

6) Mitigation - Do not establish network connections based on user-controlled data and ensure that the request is being sent to the expected destination. If user data is necessary to build the destination URI, use a level of indirection: create a list of legitimate resource names that a user is allowed to specify, and only allow the user to select from the list. With this approach the input provided by the user is never used directly to specify the resource name.

```
700                 URL base = http.getURL();
701                 String loc = http.getHeaderField("Location");
702                 URL target = null;
703                 if (loc != null) {
704                     target = new URL(base, loc);
705                 }
706                 http.disconnect();
707                 // Redirection should be allowed only for HTTP and HTTPS
708                 // and should be limited to 5 redirects at most.
709                 if (target == null || !("http".equals(target.getProtocol()) || "https".equals(target.getProtocol()))
710                         || redirects >= 5) {
711                     throw new SecurityException("illegal URL redirect");
712                 }
713                 redir = true;
714                 c = target.openConnection();
715                 redirects++;
```

Vulnerability #4:

1) Vulnerability Name - Server-Side Template Injection

2) Business Impact - It can make the attacker steal data from the business or corrupt the users data which will impact both in monetary and reputation terms.

3) Affected Component - The call to evaluate() in VelocityMessagePreparator.java on line 60 evaluates usercontrolled data as a template engine's template, allowing attackers to access the template context and in some cases inject and run arbitrary code on the applicationserver.

4) Description - Template engines are used to render content using dynamic data. This context data is normally controlled by the user and formatted by the template to generate web pages, emails and the like. Template engines allow powerful language expressions to be used in templates in order to render dynamic content, by processing the context data with code constructs such as conditionals, loops, etc.

5) Result - If an attacker is able to control the template to be rendered, they will be able to inject expressions that will expose context data or even run arbitrary commands on the server.

6) Mitigation - Whenever possible do not allow users to provide templates. If user-provided templates are necessary, perform careful input validation to prevent malicious code from being injected in the template.

```
55      @Override
56      @SuppressWarnings("unchecked")
57      public List<Template> getTemplatesByName(String name) {
58          log.info("Get template " + name);
59          return sessionFactory.getCurrentSession().createQuery("from Template as template where template.name = ?")
60                  .setString(0, name).list();
61      }
```

HibernateTemplateDAO.java:60 org.hibernate.Query.list()

```
58          try {
59              engine.evaluate(context, writer, "template", // I have no idea what this is used for
60                  template.getTemplate());
61          }
```

VelocityMessagePreparator.java:60
org.apache.velocity.app.VelocityEngine.evaluate()
Reference: Fortify Security Report