### **CSC-515 Software Security**

### **Project Part 1: OWASP Top 10**

## 

**Objective:**

Security review of OpenMRS system based upon the OWASP list of top 10 vulnerabilities,Documentation of the findings and Remediation suggestions to correct and adverse findings using repeatable black box test plan

## 

## **Team members:**

## Ashwin Anil Kumar (aanilku)

## Bhavya Bansal (bbansal)

## Neetish Pathak (npathak2)

## Rushi Bhatt (rbhatt)

## 

## **List of tools used:** Zap, sqlmap, burp suite, dependency check cli, postman

**OWASP Top 10 vulnerabilities:**

* 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:**

##### Log into openMRS website using a valid username and password. (Lets call this User1)

##### 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

##### 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

##### 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

##### **JSESSIONID:** <Stores session ID> , **\_REFERENCE\_APPLICATION\_LAST\_USER\_:** <Last user Id>, **referenceapplication.lastSessionLocation:** <Int>

##### Once you get this information, you can use this information to perform actions by faking this user

##### Open another incognito window (to simulate the attacker), and go to the openMrs URL

##### 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

##### 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:**

##### 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

##### 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.

#### **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

#### **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.

### **References**

1. <https://blog.smartbear.com/apis/api-security-testing-how-to-hack-an-api-and-get-away-with-it-part-2-of-3/>
2. <https://www.owasp.org/index.php/OWASP_Dependency_Check>
3. <https://www.owasp.org/index.php/Top_10_2017-Top_10>
4. <https://developers.google.com/web/tools/chrome-devtools/manage-data/cookies>