Audit Logging/ Implementation, Static Analysis with Fortify, Fuzzing with ZAP, Client-side bypassing with ZAP, Security Requirements

Project Phase III

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

0. Module Selection:

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

1. Audit/Logging Implementation:

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

Module: Find/ Create Patient Module

1. Adding Allergy

TESTCASE ID: OpenMRS_Log1

Steps:

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_ Log1	Type of Access: CREATE User: System Administrator/ Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not to be Logged: Values for Allergies	Type of Access: CREATE User: Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not Logged: Values for Allergies	PASS

2. Update Allergy

TESTCASE ID: OpenMRS_Log2

Steps:

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_ Log2	Type of Access: EDIT User: System Administrator/ Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not to be Logged: Values for Allergies .	Type of Access: EDIT User: Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not Logged: Values for Allergies	PASS

3. Viewing Sensitive Medical Data
TESTCASE ID: OpenMRS_Log3

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_ Log3	Type of Access: VIEW User: System Administrator/ Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not to be Logged: Values for Allergies, Diagnosis, Vitals .	Type of Access: VIEW User: Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not Logged: Values for Allergies, Diagnosis, Vitals.	PASS

4. Removing Allergy
TESTCASE ID: OpenMRS_Log4

Steps:

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS

OpenMRS_ Log4	Type of Access: DELETE User: System Administrator/ Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not to be Logged: Values for Allergies.	Type of Access: DELETE User: Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not Logged: Values for Allergies..	PASS

5. Adding Diagnosis

TESTCASE ID: OpenMRS_Log5

Steps:

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS

OpenMRS_ Log5	Type of Access: CREATE User: Doctor Date/Time of access What: Diagnosis information for Patient ID 1002JN Not to be Logged: Values for Diagnosis .	Type of Access: CREATE User: Doctor Date/Time of access What: Diagnosis information for Patient ID 1002JN Not Logged: Values for Diagnosis	PASS

6. Update Diagnosis
TESTCASE ID: OpenMRS_Log6

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_ Log6	Type of Access: EDIT User: Doctor Date/Time of access What: Diagnosis information for Patient ID 1002JN Not to be Logged: Values for Diagnosis .	Type of Access: EDIT User: Doctor Date/Time of access What: Diagnosis information for Patient ID 1002JN Not Logged: Values for Diagnosis	PASS

7. Removing Diagnosis

TESTCASE ID: OpenMRS_Log7

Steps:

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_ Log7	Type of Access: DELETE User: Doctor Date/Time of access What: Diagnosis information for Patient ID 1002JN Not to be Logged: Values for Diagnosis.	Type of Access: DELETE User: Doctor Date/Time of access What: Allergies information for Patient ID 1002JN Not Logged: Values for Diagnosis.	PASS

8. Removing A Patient

TESTCASE ID: OpenMRS_Log8

Steps:

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_ Log8	Type of Access: DELETE User: System Administrator/ Super User Date/Time of access What: Medical information for Patient ID 1002JN Not to be Logged: Values for Visits, Diagnosis, Vitals.	Type of Access: DELETE User: Super User Date/Time of access What: Medical information for Patient ID 1002JN Not Logged: Values for Visits, Diagnosis, Vitals.	PASS

9. Registering A Patient
TESTCASE ID: OpenMRS_Log9

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_ Log9	Type of Access: CREATE User: System Administrator/ Super User Date/Time of access What: Medical information for new Patient Not to be Logged: Personal information of the patient like address, phone number.	Type of Access: CREATE User: Super User Date/Time of access What: Medical information for new Patient Not Logged: Personal information of the patient like address, phone number.	PASS

10. Editing Patient Info
TESTCASE ID: OpenMRS_Log10

Steps:

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

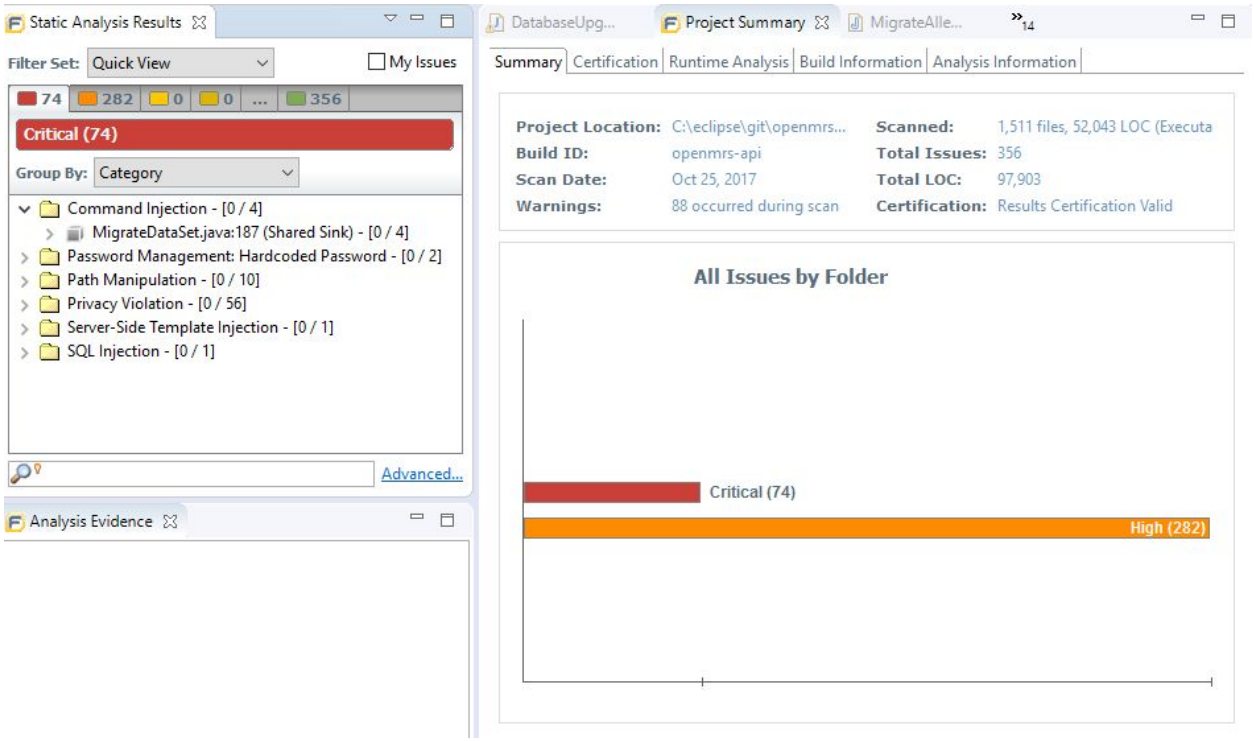
RESULTS:

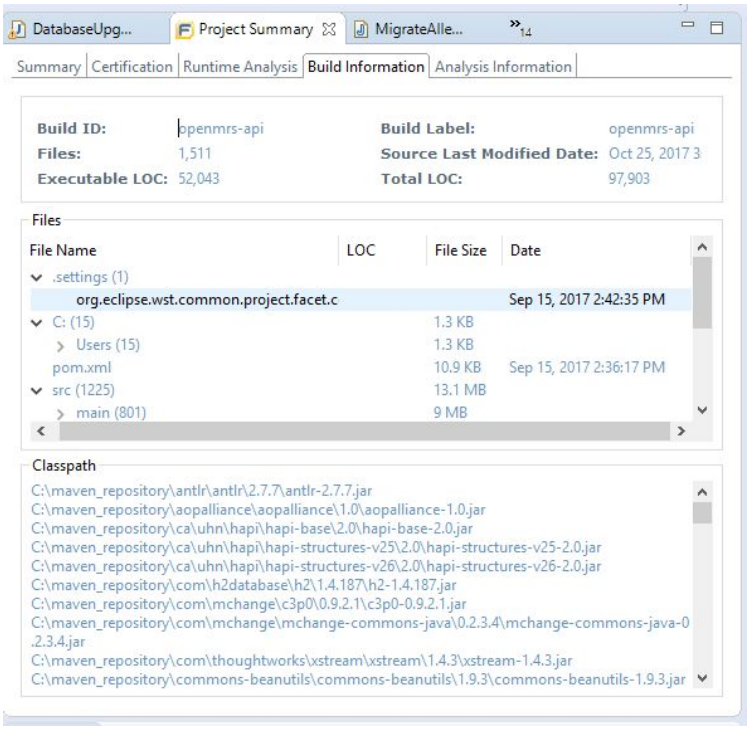
TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_ Log10	Type of Access: EDIT User: System Administrator/ Super User Date/Time of access What: Medical information for new Patient Not to be Logged: Personal information of the patient like address, phone number.	Type of Access: EDIT User: Super User Date/Time of access What: Medical information for new Patient Not Logged: Personal information of the patient like address, phone number.	PASS

Adequacy of Logging In OpenMRS

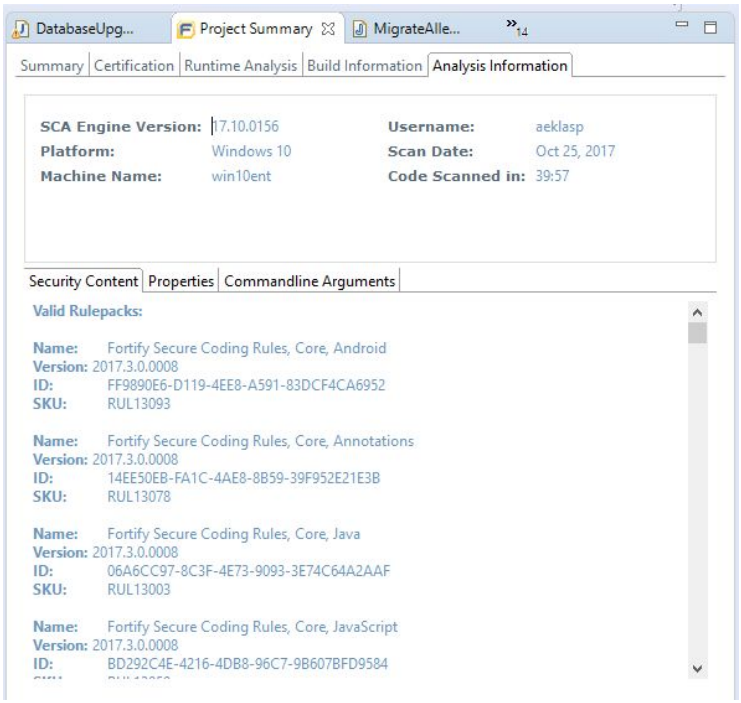
OpenMRS uses Tomcat logs to register its logging. From the test cases above, we can see that the necessary events are logged with appropriate details.
It logs the users and the actions whenever data is added/modified/deleted/viewed along with the timestamp, but does not log the sensitive medical data.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

2. Static analysis with Fortify





Build Summary



Analysis summary

Issues Identified in OpenMRS-API with Fortify

1. **Privacy Violation**

   Issue:

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

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

   Change suggested:

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

2. **Command Injection**

Issue:

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

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

Change suggested:

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

3. SQL Injection

Issue:

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

On line **164 of MigrateAllergiesChangeSet.java,the methodgetConceptByGlobalProperty(),** invokes a SQL query which is dynamically built using input coming from the user (untrusted source). This makes the code vulnerable to a SQL injection attack, when the attacker can modify the dynamically generated SQL query to execute malicious SQL commands.

Change suggested:

- Validate the user input parameters (“uuid”) with a whitelist or blacklist, Use parametrized SQL commands to disallow data-directed context changes and preventing nearly all SQL injections. Construct parameterized SQL commands

using regular SQL and use the user supplied as bind parameters in them, this way the database can know which part should be treated as a command and which part should be treated as data.

- Have Stored procedures and make calls to them with validated user input, or use modern web frameworks like Spring or Struts which do user validation.

4. Path Manipulation

Issue:

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

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

Change suggested:

Validate the parameters sent to File command by verifying if the user or application have access to those directories. Or run the application under user who does not have access to other directories (critical) at system level.

Another suggestion is to have a another level of indirection by creating a list of legitimate resource names that a user is allowed to specify, and only allow the user to select from the list.

5. Password Management: Hardcoded password

Issue:

"String password = "1d1436658853acecead72e92f1ae9089a0000fbb38cea519ce34eae9f28523930ecb212177dbd607d83dc275fde3e9ca648deb557d503ad0bcd01a955a394b2"; "

"public static String SCHEDULER_DEFAULT_PASSWORD = "test";"

At line **58 in SecurityTest.java , function hashMatches_shouldMatchStrings HashedWithSha512AlgorithmAnd128CharactersSalt()** uses a hardcoded password.

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

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

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

6. Log Forging

Issue:

104: *“log.warn("request for username '" + username + "' not found");”*
329: *“log.info("updating password for " + u.getUsername());”*

The method getUserByUsername() in HibernateUserDAO.java writes unvalidated user input to the log on line 104. The method changePassword() in HibernateUserDAO.java writes unvalidated user input to the log on line 329.

An attacker could take advantage of this behavior to forge log entries or inject malicious content into the log, which may affect conclusions of future log analysis or forensic analysis.

Change Suggested:
Log forging attacks can be mitigated by adding 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 like the code here does (username), verify its validity with server controlled values and log the server controlled values.

7. Password Management: Password in Configuration file

Issue:

“admin_user_password=Admin123”
“ connection.password=sa”

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

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

Change Suggested:

Encrypt(or Hash with a salt) the passwords before storing them in the configuration files and decrypt then at runtime. For first time users generate a random password at installation and store its encrypted form in the configuration file. Another option is to have the entire configuration file encrypted and decrypted at runtime. For system passwords, passwords can be securely set as environment variables and assessed appropriately.

8. Privacy Violation: Heap Inspection

Issue:

“pass = new String(password);”

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

Change Suggested:

Store sensitive information like passwords in char/byte arrays instead of String, so that they can be programmatically cleared from the memory. For example

```
private JPasswordField pf;
...
final char[] password = pf.getPassword();
// use the password
...
// erase when finished
Arrays.fill(password, ' ');
```

9. Denial of Service: Regular Expression

Issue:

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

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

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

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

10. Dynamic Code Evaluation: Unsafe XStream Deserialization

Issue:

"return (T) xstream.fromXML(serializedObject);"

At line 107 method deserialize in the file SimpleXStreamSerializer.java, unvalidated XML is deserialized, XStream library provides the developer with an easy way to transmit objects, serializing them to XML documents. But XStream can by default deserialize dynamic proxies allowing an attacker to run arbitrary Java code on the server when the proxy's InvocationHandler is invoked. This can also be used to cause a Denial of service attack by sending a very long input file, since the input is not validated

Change Suggested:
Validate the input to the serializer by using a whitelist or blacklist filter before passing to XStream.
Associate a timer with the calls to xstream and kill it if timeout occurs, this way Denial of service attacks can be prevented.
XStream implicitly prevents the deserialization of known bad classes such as java.beans.EventHandler that can be used by attackers to run arbitrary commands. In addition, starting with XStream 1.4.7, it is possible to define permissions for types. These permissions can be used to explicitly allow or deny the types that will be deserialized and

so it is not possible to inject unexpected types into an object graph. Any application that deserializes data from an external source should use this feature to limit the danger of arbitrary command execution.Always use the whitelist approach (allowed types) since many classes can be used to achieve remote code execution and to bypass blacklists.

3. FUZZING WITH ZAP:

Module Selected: Find/Create Patient

Injection:

Test Case Id: INJ_01

Instructions to execute INJ_01:

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

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

Results:

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

Test Case Id: INJ_02

Instructions to execute INJ_02:

1. Open chrome Browser.
2. Run OpenMRS

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

How the fuzzing exercise do?
Yes, found something.

Results:

Test Case ID	Expected Results	Actual Result	Result
INJ_02	The report should not show any attacks that were successful.	Report contains a command with the status “Reflected”, signalling a successful attack.	FAIL

Comments for Injection Attacks:
In INJ_01, the login form is properly input validated. Hence, no vulnerability is detected there. In INJ_02, the test case injecting “create username identified by pass123 temporary tablespace temp default tablespace users;” in the name field shows a successful injection attack because no input validation is done on the form there.

Buffer Overflow:

Test Case Id: BO_01

Instructions to execute BO_01:

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

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

Results:

Test Case ID	Expected Results	Actual Result	Result
BO_01	The report should not show any successful attacks.	No attacks are successful	PASS

Test Case Id: BO_02

Instructions to execute:BO_02

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

12. Start the fuzzer.

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

Results:

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

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

XSS:

Test Case Id: XSS_01

Instructions to execute: XSS_01

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

13. When you try to edit the record again using one of the injected values (eg. <script>alert('xss')</script>), a popup box appears.

How the fuzzing exercise do?
Yes, found something.

Results:

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

Test Case Id: XSS_02

Instructions to execute: XSS_02

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

How the fuzzing exercise do?
Yes, found something.

Vulnerabilities exist. Entering <DIV STYLE="background-image: url(http://hackers.org/xss.js)">Div Body</DIV> in the role field shows a successful XSS attack.

Results:

Test Case ID	Expected Results	Actual Result	Result
XSS_02	The report should not show any attacks that were successful.	Report contains a command with the status “Reflected”, signalling a successful attack.	FAIL

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

SQL Injection Attack:

Test Case Id: SQL_INJ_01

Instructions to execute SQL_INJ_01:

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

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

Results:

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

Test Case Id: SQL_INJ_02

Instructions to execute SQL_INJ_02:

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

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

Results:

Test Case ID	Expected Results	Actual Result	Result

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

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

4. Client-Side Bypassing with ZAP:

Module Selected: Find/Create Patient

Instructions to Set up ZAP and OpenMRS:

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

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

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

Address: localhost Port: 8008

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

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

6. After OpenMRS starts, navigate to the OpenMRS instance at:

“http://localhost:8081/openmrs-standalone “ in the Chrome browser.

This should now pop up in the “Sites” list in ZAP.

1. Login Injection using ZAP

TEST CASE ID: CSBZAP_1

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

INSTRUCTIONS TO EXECUTE CSBZAP_1:

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

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

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

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

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

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

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

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

8. In ZAP, modify the POST request (Under the break Tab on the right) as follows:
   Existing:
   username=admin&password=Admin123&sessionLocation=6&redirectUrl=%2Fopenmrs-standalone%2Freferenceapplication%2Fhome.page
   Modified:
   username=admin&password=1’or’1’=’1’;--&sessionLocation=6&redirectUrl=%2Fopenmrs-standalone%2Freferenceapplication%2Fhome.page
9. Hit the Play icon on the ZAP menu bar (Right above the “Response” tab). Check for any errors on the login page.

DOCUMENTATION:

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

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

RESULTS:

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

2. XSS using ZAP

TEST CASE ID: CSBZAP_2

DESCRIPTION: CSBZAP_2 tries a cross site scripting attack.

INSTRUCTIONS TO EXECUTE CSBZAP_2:

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

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

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

Filler Information:

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

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

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

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

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

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

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

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

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

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

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

DOCUMENTATION:

PAGE URL:

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

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

RESULTS:

TEST CASE ID	**EXPECTED RESULTS**	**ACTUAL RESULTS**	**TEST CASE RESULTS**

CSBZAP_2	The javascript is not executed and the page loads normally.	The javascript is not executed and the page loads normally.	PASS

3. Stealing cookie information through XSS using ZAP

TEST CASE ID: CSBZAP_3

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

INSTRUCTIONS TO EXECUTE CSBZAP_3:

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

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

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

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

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

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

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

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

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

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

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

Existing:

GET
http://localhost:8081/openmrs-standalone/registrationapp/registerPatient/submit.action?a
ppId=referenceapplication.registrationapp.registerPatient&&givenName=Alex&middleNa
me=&familyName=Ferguson&preferred=true&gender=M&unknown=false&birthdateDay=
10&birthdateMonth=4&birthdateYear=1990&birthdate=1990-4-10&address1=123+Test+
Dr&address2=&cityVillage=Test&stateProvince=Test&country=Test&postalCode=27606
&phoneNumber=1231231234&relationship_type=8d91a210-c2cc-11de-8d13-0010c6dffd
0f-A&other_person_uuid= HTTP/1.1

Modified:

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

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

DOCUMENTATION:

PAGE URL:

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

User Field Input	**Initial Input**	**Malicious Input**
Phone Number	1231231234	%3cscript%3ealert(document.cookie)%3c/script%3e

RESULTS:

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

4. Overflow using ZAP

TEST CASE ID: CSBZAP_4

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

INSTRUCTIONS TO EXECUTE CSBZAP_4:

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

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

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

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

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

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

> http://localhost:8081/openmrs-standalone/registrationapp/registerPatient/submit.action\?appId\=referenceapplication\.registrationapp\.registerPatient ”.

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

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

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

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

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

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

DOCUMENTATION:

PAGE URL:

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

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

RESULTS:

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

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

TEST CASE ID: CSBZAP_5

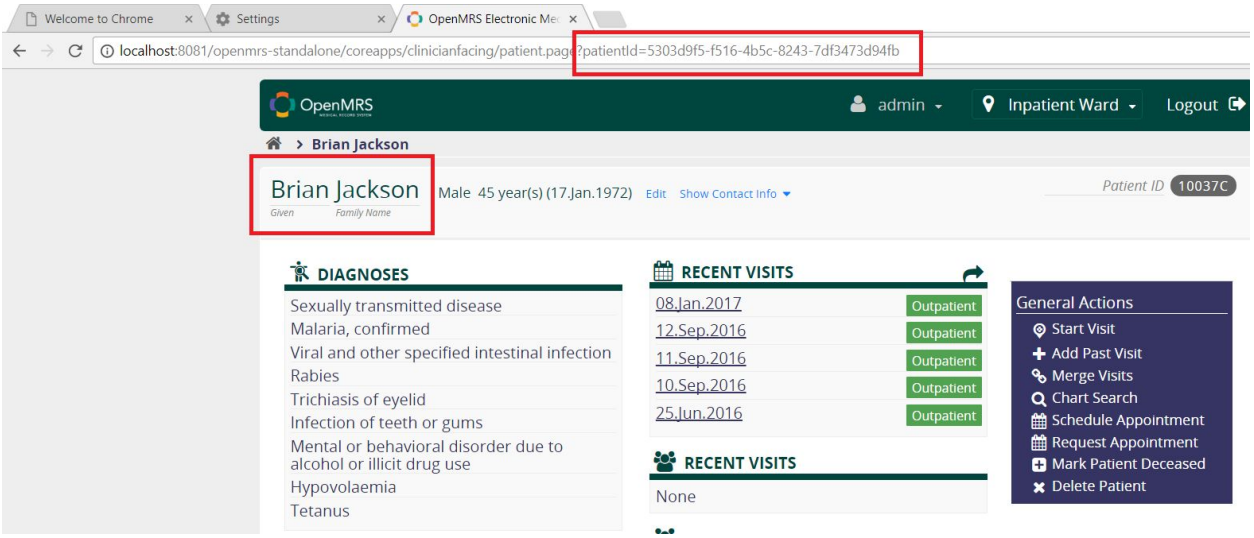
DESCRIPTION: This test is to determine whether the direct objects in the URL could be misused for executing scripts.

INSTRUCTIONS TO EXECUTE CSBZAP_5:

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

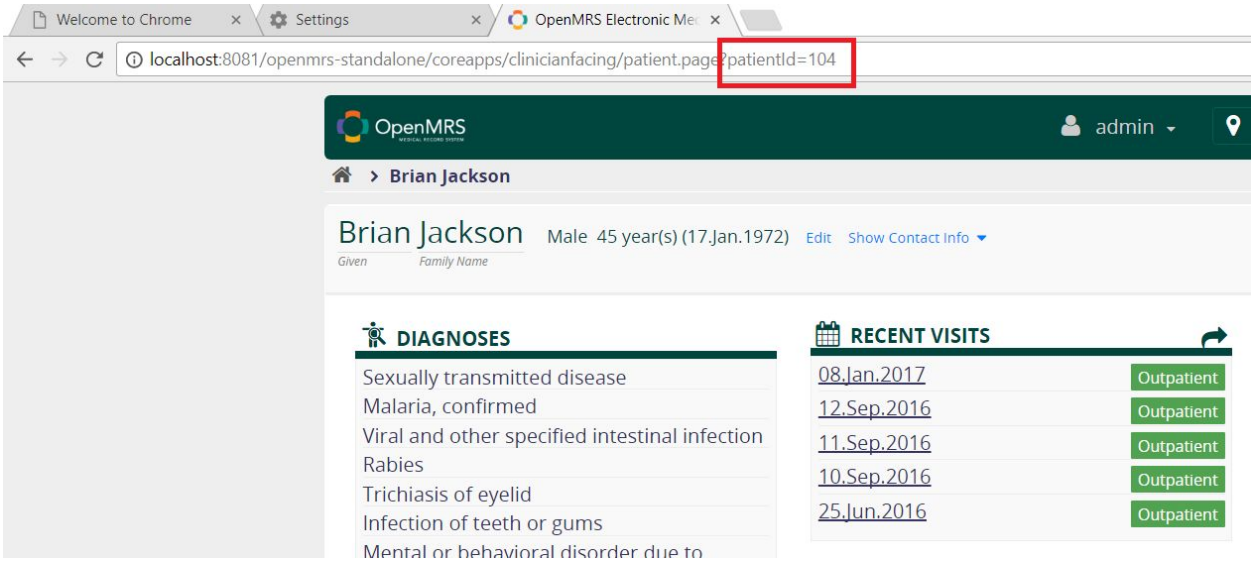
2. Log into OpenMRS (use the default username/password: admin/Admin123 and select any location) with ZAP running so that the sites tab in ZAP is populated with the login data scripts.

3. Once on the Home page, navigate to “Find a Patient” and search for “Brian Jackson” and click on the patient “Brian Jackson” from the search result. The page looks like:



Please take note of the Patient ID in the URL link.

4. Now click on the name Brian Jackson (as highlighted in the above image), upon which the page will refresh and now shows the patient’s internal ID in the URL. The screenshot looks like:



4. In ZAP, open the http://localhost:8081 site menu under the Sites tab on the left side;navigate down the menu as follows: http://localhost:8081 -> openmrs-standalone ->coreapps -> clinicianfacing-> GET:patient.page(patiendId).

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

http://localhost:8081/openmrs-standalone/coreapps/clinicianfacing/patient.page\?patientId

Then hit save to confirm the breakpoint.

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

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

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

Existing: GET
GET
http://localhost:8081/openmrs-standalone/coreapps/clinicianfacing/patient.page?patientId=5303d9f5-f516-4b5c-8243-7df3473d94fb HTTP/1.1
Modified: GET
GET
http://localhost:8081/openmrs-standalone/coreapps/clinicianfacing/patient.page?patientId=%3cscript%3ealert(document.cookie)%3c/script%3e HTTP/1.1

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

DOCUMENTATION:

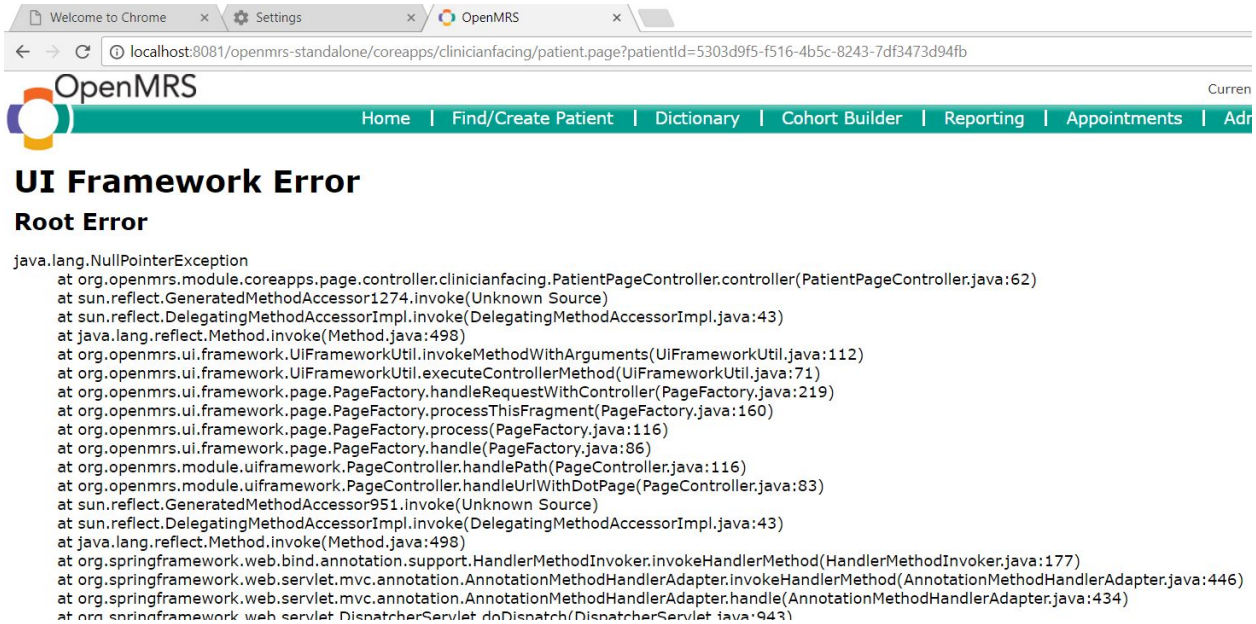
PAGE URL:
http://localhost:8081/openmrs-standalone/registrationapp/registerPatient.page?appId=referenceapplication.regisatrationapp.registerPatient

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

RESULTS:

TEST CASE ID	**EXPECTED RESULTS**	**ACTUAL RESULTS**	**TEST CASE RESULTS**
CSBZAP_5	The page should reject the script and should load the patient’s detail	The application throws an error which shows the stack trace.	PARTIAL PASS - Since, the script was not executed,

			but stack trace is shown (which is extra information - Refer the Image below)



5. Security Requirements :

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

1. OMRS_SR1:
All requests made to the pages under the “Find/Create Patient” module should use HTTPS and contain unique tokens to safeguard against CSRF attacks. This can be optionally implemented in other modules as well to secure the application as a whole .

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

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

4. OMRS_SR4:
.Error messages must show the least amount of information as needed. The stack trace leading up to the error should definitely not be printed.

5. OMRS_SR5:
Audit logging of sensitive medical data (that includes Protected Health Information(PHI)) should adhere to the “Privacy Rule” defined in the “Health Insurance Portability and Accountability Act”. This should be followed in all the modules.

6. OMRS_SR6:
Only users with proper authorization should be able to create/view patient records. For this, roles for different users must be established.

7. OMRS_SR7:
User input fields must be properly validated to prevent injection and XSS attacks. In the find/create patient module those would be the fields used to search for an existing patient and those used for entering details for new patients.

8. OMRS_SR8:

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

9. OMRS_SR9:
All users(Doctors/Nurses/Admins) passwords must be hashed and saved in the database so that the passwords can be protected against rainbow table attacks. This should be implemented in the Log-In module.

10. OMRS_SR10:
More than one level approval must be required to delete a patient's record. Thus, even if one of the user Ids is compromised the attacker won't be able to delete a record without the approval of the other authorized user. This should be implemented in Manage Patient sub-module under Administration module.