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

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

0. Module Selection:

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

1. Audit/logging implementation:

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

1) Delete Patient:

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

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

2) Add Allergy:

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

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

3) Update/Edit Allergy:

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

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG3	Update/Edit Allergy record event for patient should be logged.	INFO - LoggingAdvice.invoke(115) |2018-10-26 17:50:47,800| In method UserService.saveUser. Arguments: User=admin, INFO - LoggingAdvice.invoke(155) |2018-10-26 17:50:47,800| Exiting method saveUser	Inadequate (Generic saveUser method is logged, which makes difficult to identify this update event)

4) Remove Allergy:

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

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG4	Remove Allergy record event for patient should be logged.	INFO - LoggingAdvice.invoke(115) |2018-10-26 18:50:21,600| In method UserService.saveUser. Arguments: User=admin, INFO - LoggingAdvice.invoke(155) |2018-10-26 18:50:21,646| Exiting method saveUser	Inadequate (Method name logged is very vague, looks like the admin is saving something, nothing is logged about removing the allergy.

5) Start Visit Failed:

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

Test Case ID	Expected Result	Actual Result	Adequacy

OpenMRS_LOG5	Start Visit event for patient in case of error or exception should be logged.	Stack Trace for Exception Caused: Caused by: java.lang.IllegalArgumentException: Location does not support visits at org.openmrs.module.emrapi.adt.AdtServiceImpl.getLocationThatSupportsVisits(AdtServiceImpl.java:401) at org.openmrs.module.emrapi.adt.AdtServiceImpl.buildVisit(AdtServiceImpl.java:384) at org.openmrs.module.emrapi.adt.AdtServiceImpl.ensureVisit(AdtServiceImpl.java:264) at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source) at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source) at java.lang.reflect.Method.invoke(Unknown Source)	Adequate (Stack Trace helped finding the reason behind caused exception)

6) View Patient Record:

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

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG6	Viewing patient’s record event should be logged including patient id, user id, timestamp.	INFO - LoggingAdvice.invoke(115) |2018-10-29 00:07:40,212| In method UserService.saveUser. Arguments: User=admin, INFO - LoggingAdvice.invoke(155) |2018-10-29 00:07:40,227| Exiting method saveUser INFO - SerializationServiceImpl.getDefaultSerializer(71) |2018-10-29 00:07:41,774| No default serializer specified - using builtin SimpleXStreamSerializer.	Inadequate (Generic logging on action which doesn’t include which patient’s record is viewed. It doesn’t even log that the patient’s record is view)

7) Register Patient:

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

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG7	Registering new patient event should be logged efficient, including assigned patient id.	INFO - LoggingAdvice.invoke(115) |2018-10-26 19:33:54,604| In method PatientService.savePatient. Arguments: Patient=Patient#null, INFO - LoggingAdvice.invoke(155) |2018-10-26 19:33:54,651| Exiting method savePatient	Inadequate (Logs is insufficient to identify which patient was registered)

8) Edit Patient:

Steps:
a) Login to OpenMRS portal through http://localhost:8082/openmrs-standalone/
Using credentials user = “admin” and password = “Admin123” and select any location.
b) Click on “Find Patient Record” and search using query “10010W” and click on the returned patient’s record.
c) On the top click on “Edit”, change any personal information, and save form by confirming on the popped up message.
d) Check the log file at location “C:\openMRS\referenceapplication-standalone-2.8.0\appdata\openmrs”

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG8	Edit Patient event should be logged including patient id, user, timestamp details.	INFO - LoggingAdvice.invoke(115) |2018-10-29 00:17:48,970| In method PatientService.savePatient. Arguments: Patient=Patient#37, INFO - LoggingAdvice.invoke(155) |2018-10-29 00:17:49,642| Exiting method savePatient INFO - LoggingAdvice.invoke(115) |2018-10-29 00:17:50,127| In method UserService.saveUser. Arguments: User=admin, INFO - LoggingAdvice.invoke(155) |2018-10-29 00:17:50,174| Exiting method saveUser	Adequate (Patient ID, Timestamp, user details are logged sufficiently)

9) Request Appointment:

Steps:
a) Login to OpenMRS portal through http://localhost:8082/openmrs-standalone/
Using credentials user = “admin” and password = “Admin123” and select any location.
b) Click on “Find Patient Record” and search using query “10010W” and click on the returned patient’s record.
c) Click on “Request Appointment”, and enter the mandatory details and save.

d) Check the log file at location “C:\openMRS\referenceapplication-standalone-2.8.0\appdata\openmrs”

Test Case ID	Expected Result	Actual Result	Adequacy
OpenMRS_LOG9	Requested Appointment event should be logged with timestamp, uuid.	INFO - LoggingAdvice.invoke(115) |2018-10-26 19:51:45,326| In method AppointmentService.saveAppointmentRequest. Arguments: AppointmentRequest=AppointmentRequest[hashCode=71a2021c,uuid=57b2e640-7363-4fff-b0ff-7cb36ba3a415], INFO - LoggingAdvice.invoke(155) |2018-10-26 19:51:45,342| Exiting method saveAppointmentRequest INFO - LoggingAdvice.invoke(115) |2018-10-26 19:51:46,990| In method UserService.saveUser. Arguments: User=admin, INFO - LoggingAdvice.invoke(155) |2018-10-26 19:51:46,990| Exiting method saveUser	Adequate (User details are logged along with appointment request id)

10) Merge Visits:

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

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

2. Static analysis with Fortify OR Coverity:

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

Fortify Analysis:

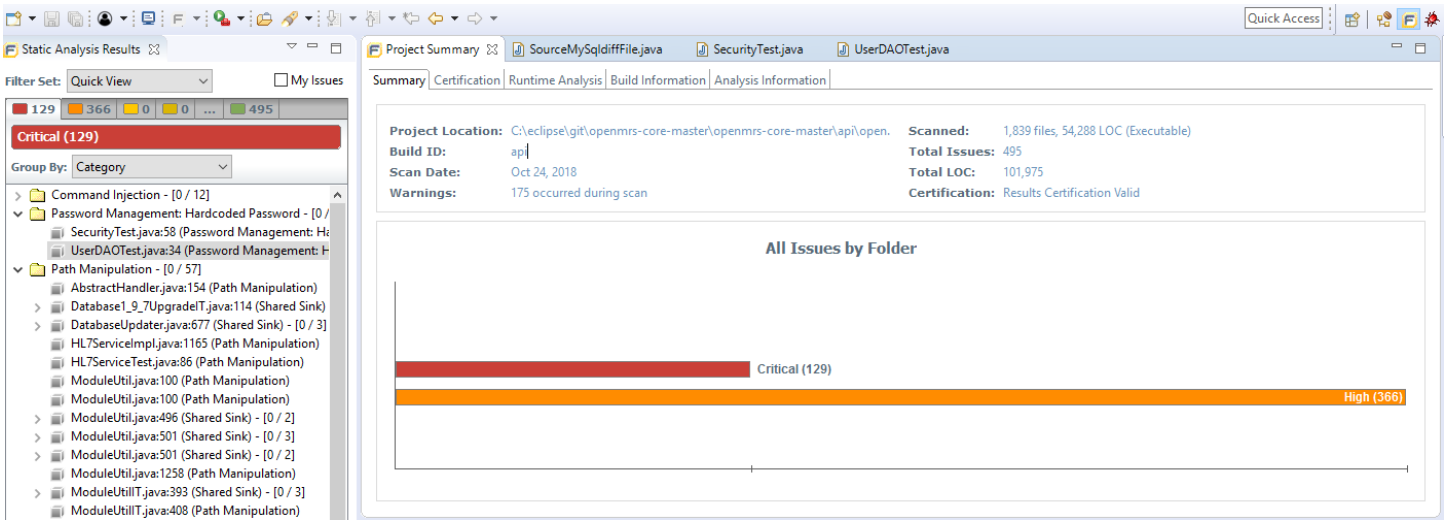


Figure: Summary for Fortify run on openmrs-api

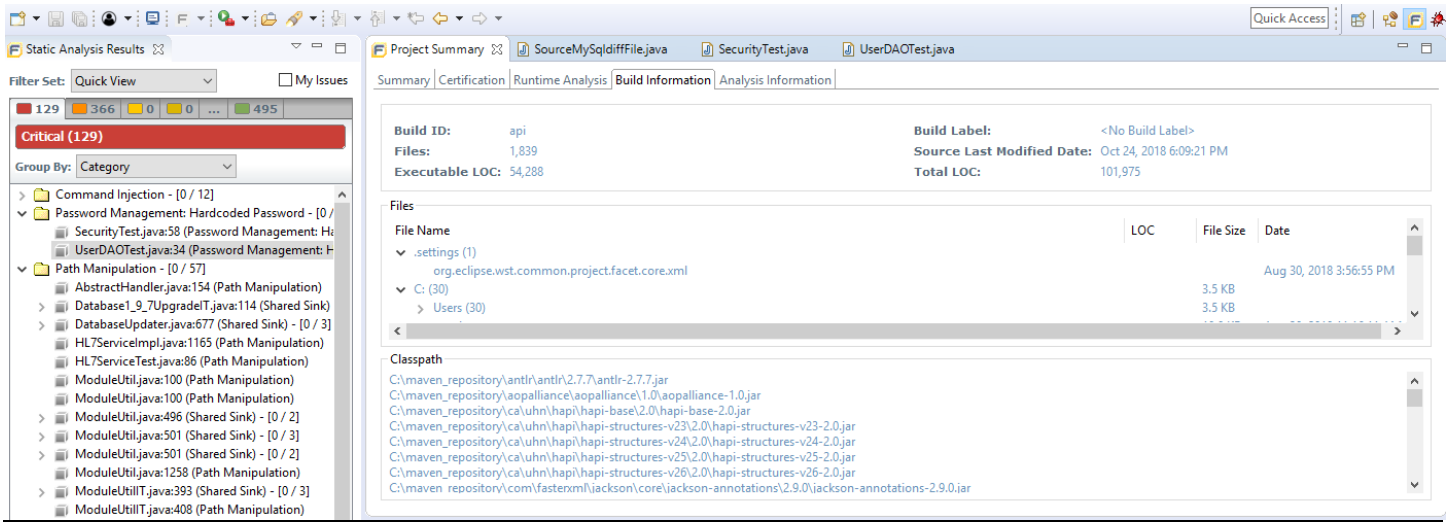


Figure: Build Information for Fortify run

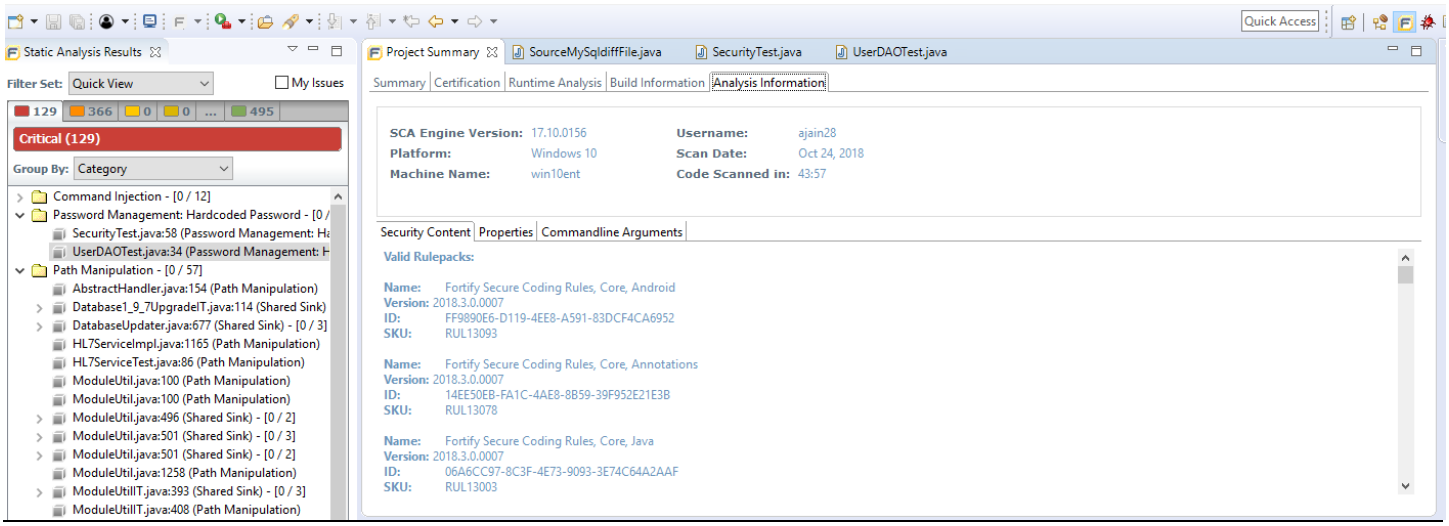


Figure: Analysis for Fortify run

Issue found in OpenMRS using Fortify tool:

1. Command Injection:

Possible Problem:

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

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

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

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

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

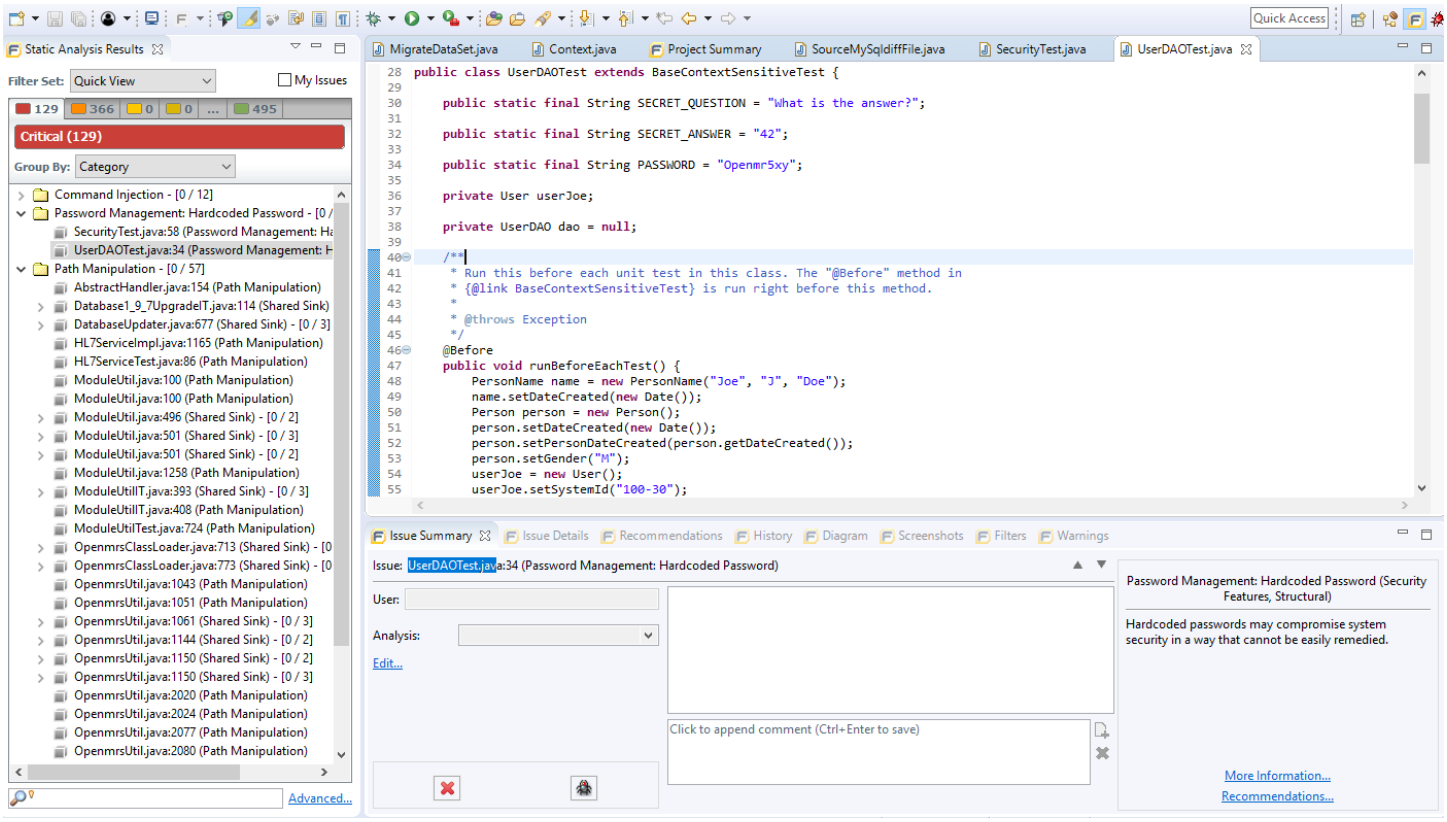
2. Password Management: Hardcoded Password (Security Features, Structural)

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

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

It is never a good idea to hardcode a password. Not only does hardcoding a password allow all of the project's developers to view the password, it also makes fixing the problem extremely difficult. Once the code is in production, the password cannot be changed without patching the software. If the account protected by the password is compromised, the owners of the system will be forced to choose between security and availability.

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



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

3. Privacy Violation:

Possible Problem:

The method authenticate() in Context.java mishandles confidential information, which can compromise user privacy and is often illegal.
Privacy violations occur when Private user information enters the program. Or the data is written to an external location, such as the console, file system, or network.
In this case the data is passed to debug() in Context.java at line 293.

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

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

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

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

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

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

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

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

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

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

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

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

MigrateAllergiesChangeSet.java at line 165

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

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

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

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

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

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

Possible Problem:
Hardcoded passwords may compromise system security in a way that cannot be easily remedied.It is never a good idea to hardcode a password. Not only does hard coding a password allow all of the project's developers to view the password, it also makes fixing the problem extremely difficult. Once the code is in production, the

password cannot be changed without patching the software. If the account protected by the password is compromised, the owners of the system will be forced to choose between security and availability.

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

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

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

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

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

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

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

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

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

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

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

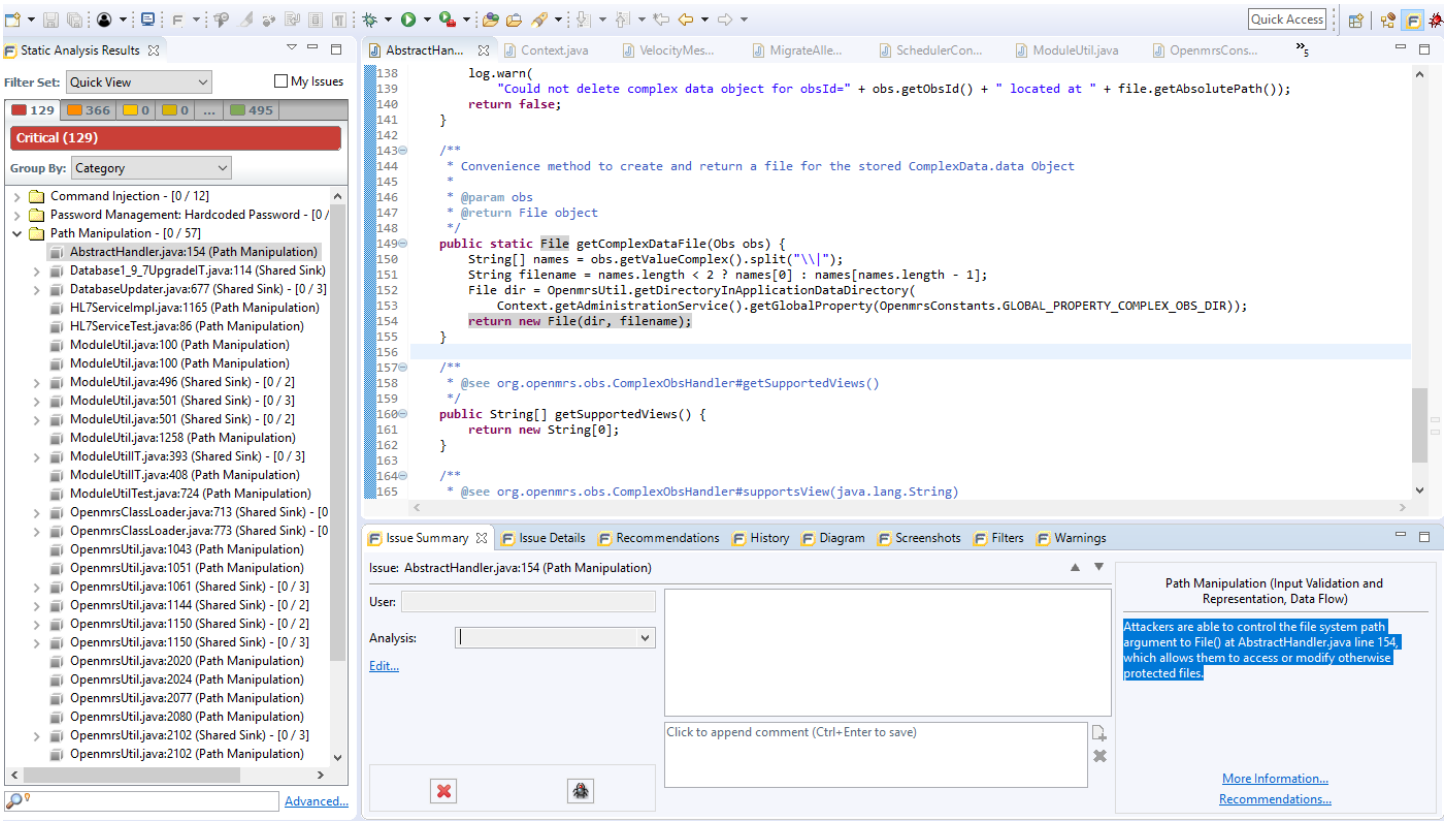
Possible Problem:
Attackers are able to control the file system path argument to File() at AbstractHandler.java line 154, which allows them to access or modify otherwise protected files. Path manipulation errors occur when an attacker is able to specify a path used in an operation on the file system.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

In this case, the attacker may specify the value that enters the program at get() in HibernateObsDAO.java at line 73, and this value is used to access a file system resource at File() in AbstractHandler.java at line 154.

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

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

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



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

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

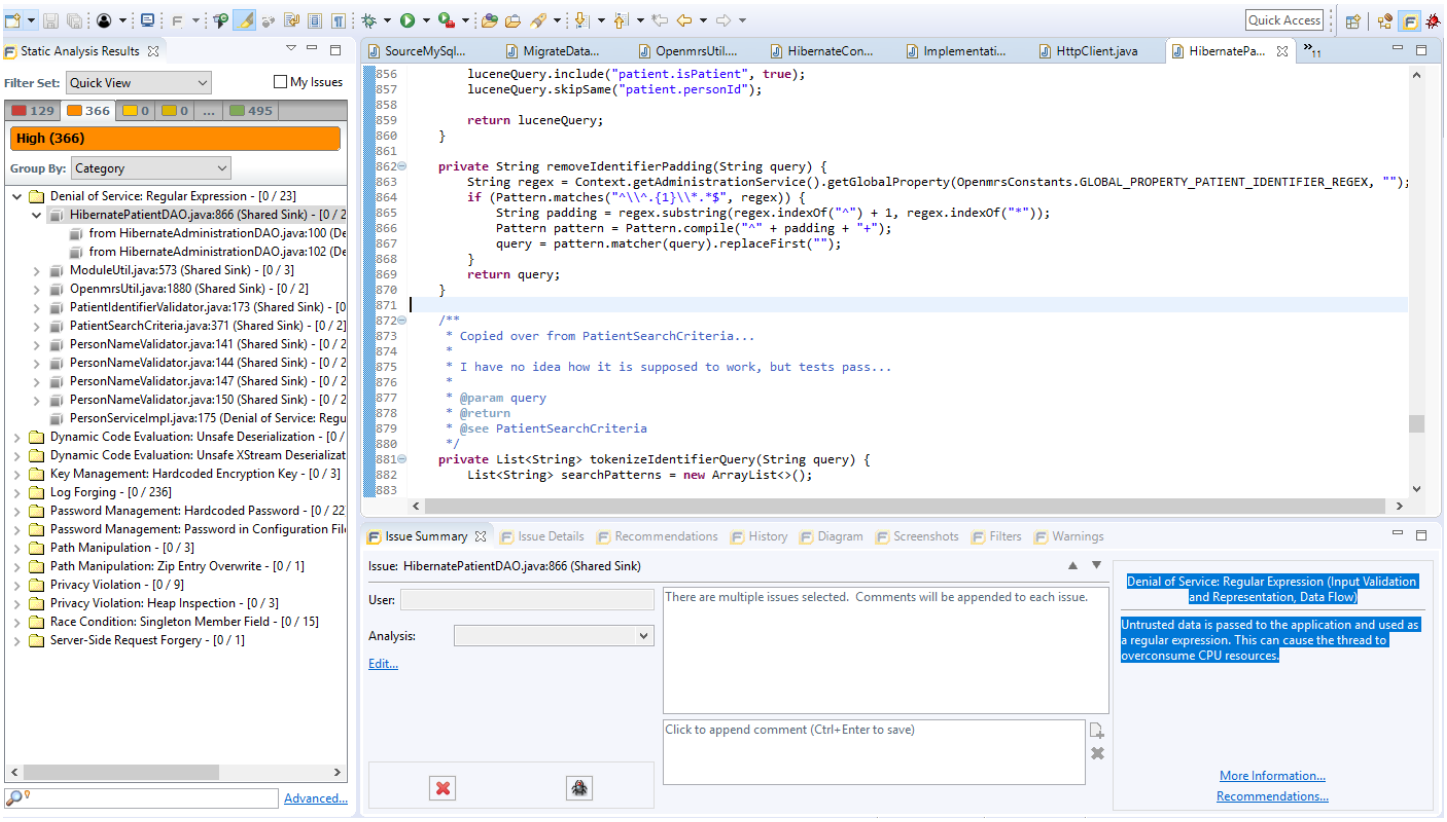
Possible Problem:

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

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

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

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



Weakness mitigated by changes:
This issue handle DDOS attack.

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

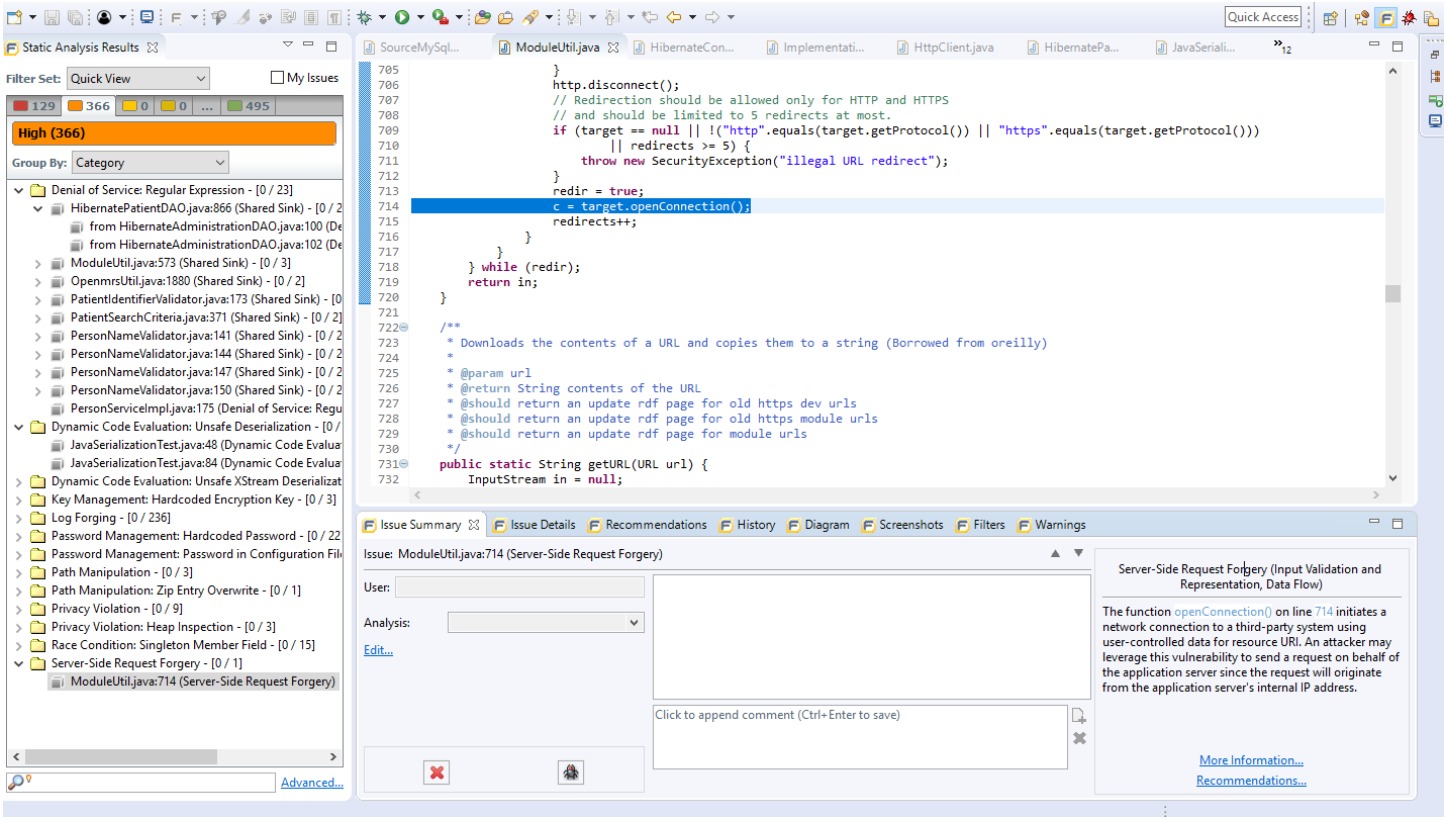
Possible Problem:

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

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

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

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



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

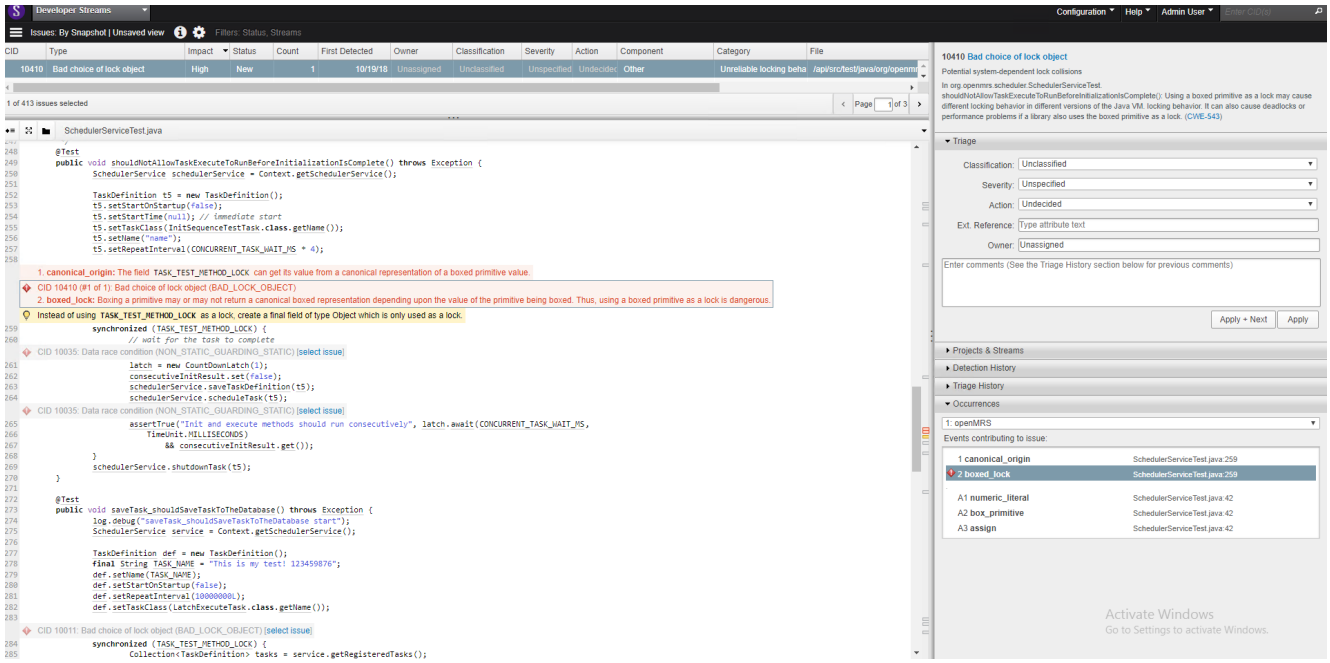
Issue found in OpenMRS using Coverity tool:

1. Bad choice of lock object

Problem Description:
It happens in org.openmrs.scheduler.SchedulerServiceTest.
shouldNotAllowTaskExecuteToRunBeforeInitializationIsComplete(): Here a boxed primitive as a lock is used which shsows in variance in locking behaviour across different versions of Java VM, and it can also cause deadlocks or performance problems if a library also uses the boxed primitive as a lock. (CWE-543)

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

Reference to report:



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

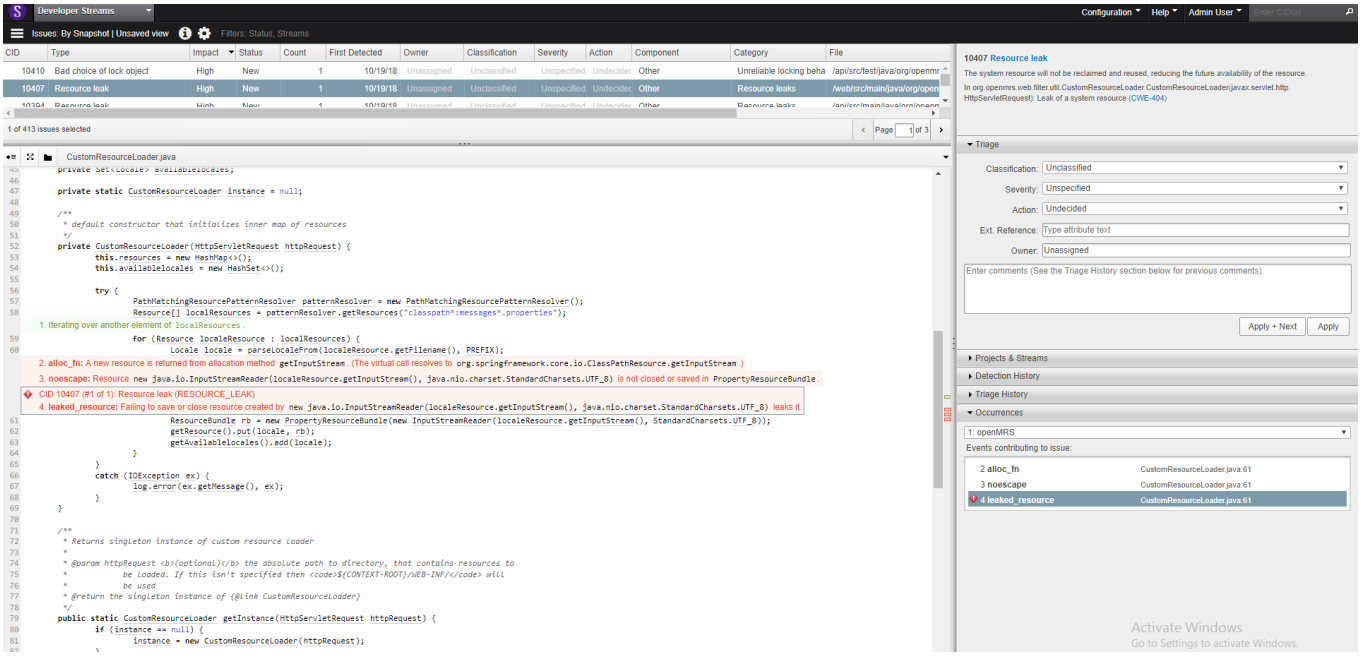
2. Resource leak

Problem Description:

It happens in org.openmrs.web.filter.util.CustomResourceLoader.CustomResourceLoader(javax.servlet.http.HttpServletRequest)
So in that file, system resource was not reclaimed and reused, reducing the future availability of the resource.Leak of a system resource (CWE-404)

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

Reference to report:



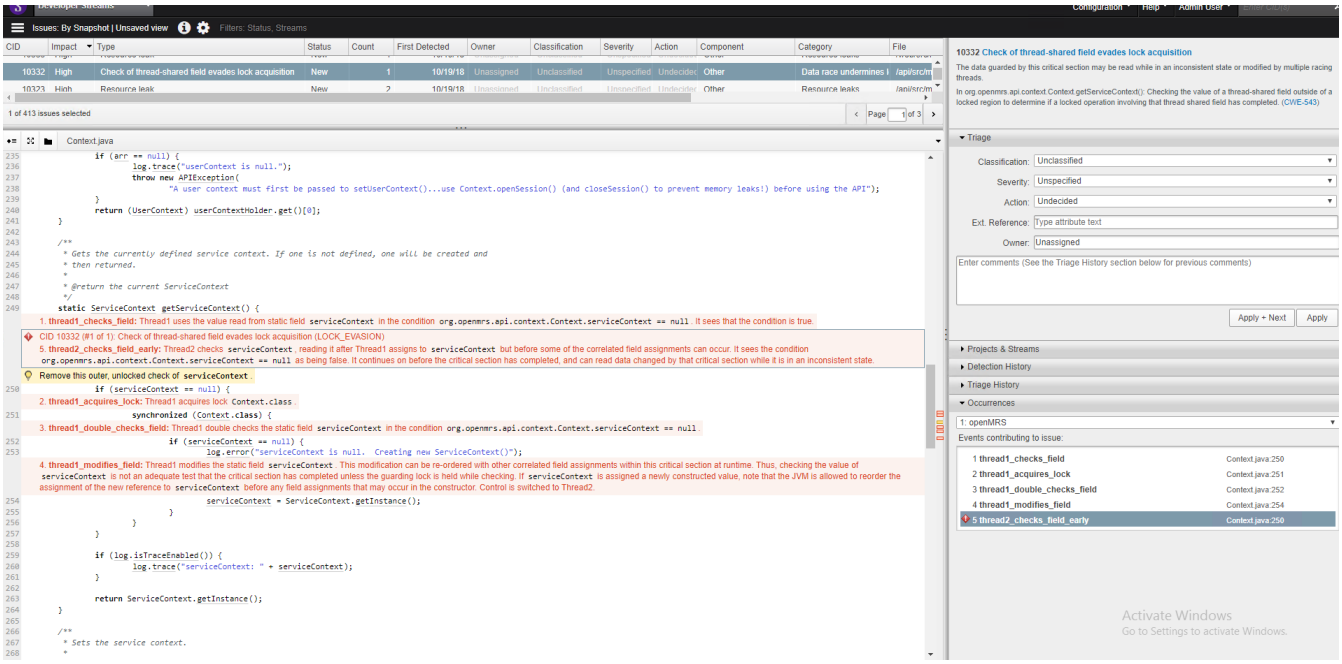
Weakness mitigated by changes:
Changes will help reducing the fragmentation of system resources, and will help reclaiming the unused system resources.

3. Check of thread-shared field evades lock acquisition

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

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

Reference to report:



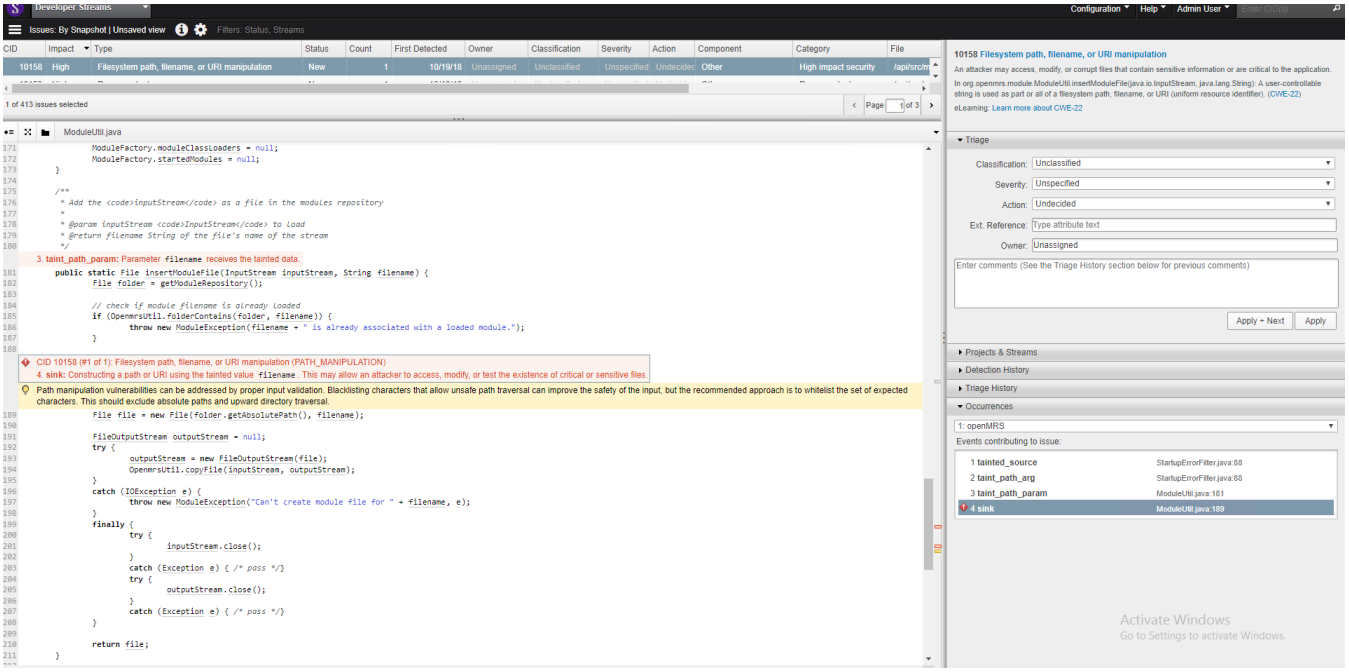
Weakness mitigated by changes:
Suggested change will mitigate the race condition as exhibited above, and will help prevent evasion of lock acquisition.

4. Filesystem path, filename, or URI manipulation

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

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

Reference to report :



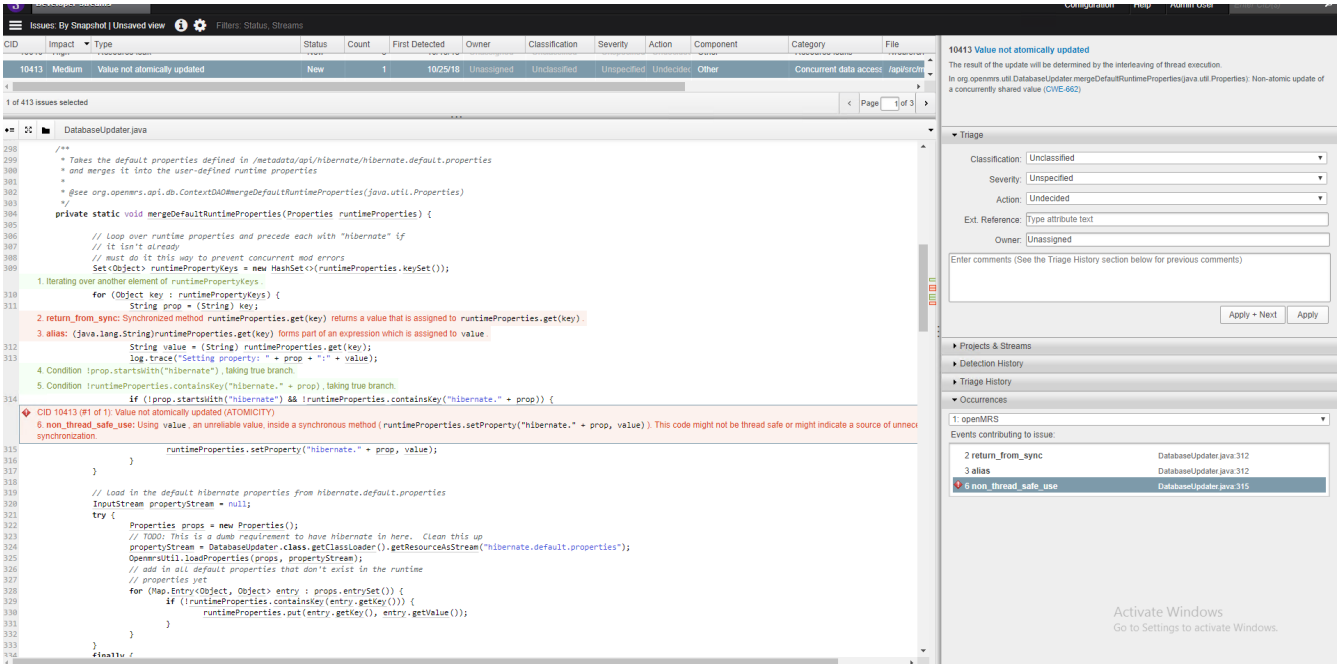
Weakness mitigated by changes:
Suggested changes will help in excluding absolute paths and upward directory traversal,thus mitigating malicious manipulation of filename or path.

5. Value not atomically updated

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

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

Reference to report:



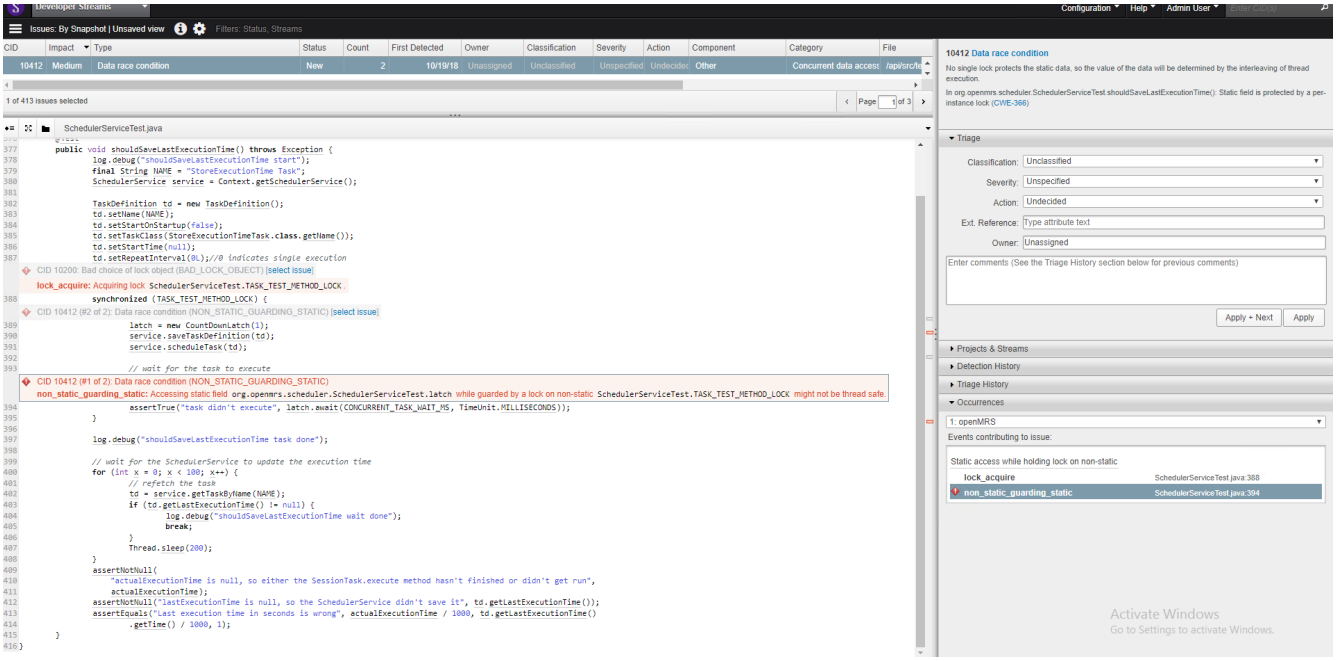
Weakness mitigated by changes:
Unreliable updates can be mitigated,by synchronising the transactions in atomic manner.

6. Data race condition

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

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

Reference to report:



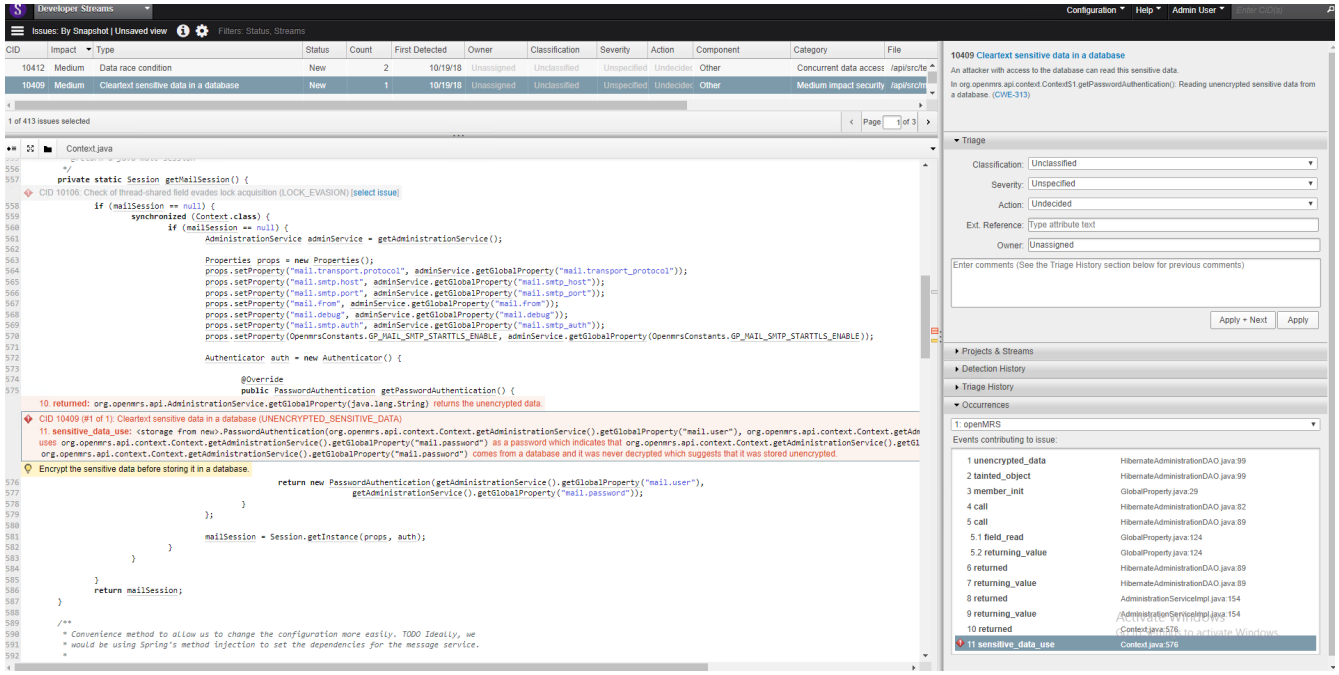
Weakness mitigated by changes:
Suggested changes will help mitigate the resource locking issues in case of interleaving thread execution.

7. Cleartext sensitive data in a database

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

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

Reference to report:



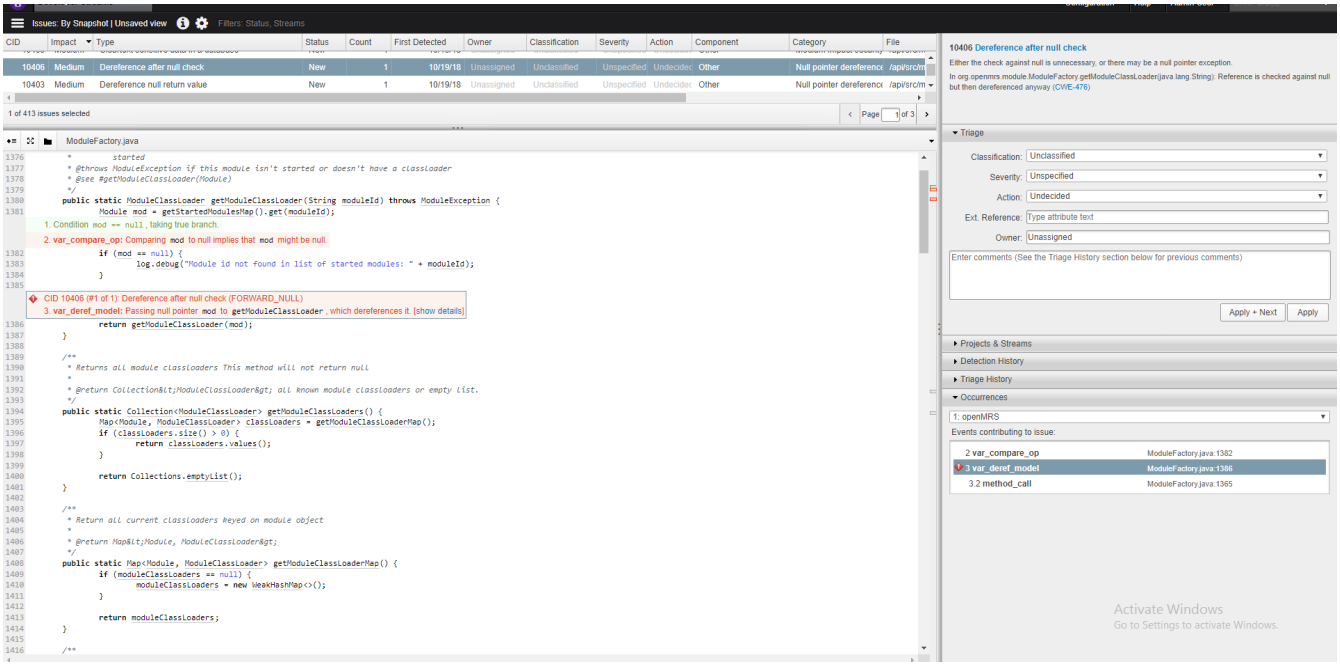
Weakness mitigated by changes:
Changes will help prevent the revelation of sensitive data in case of attacks done on database.

8. Dereference after null check

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

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

Reference to report:



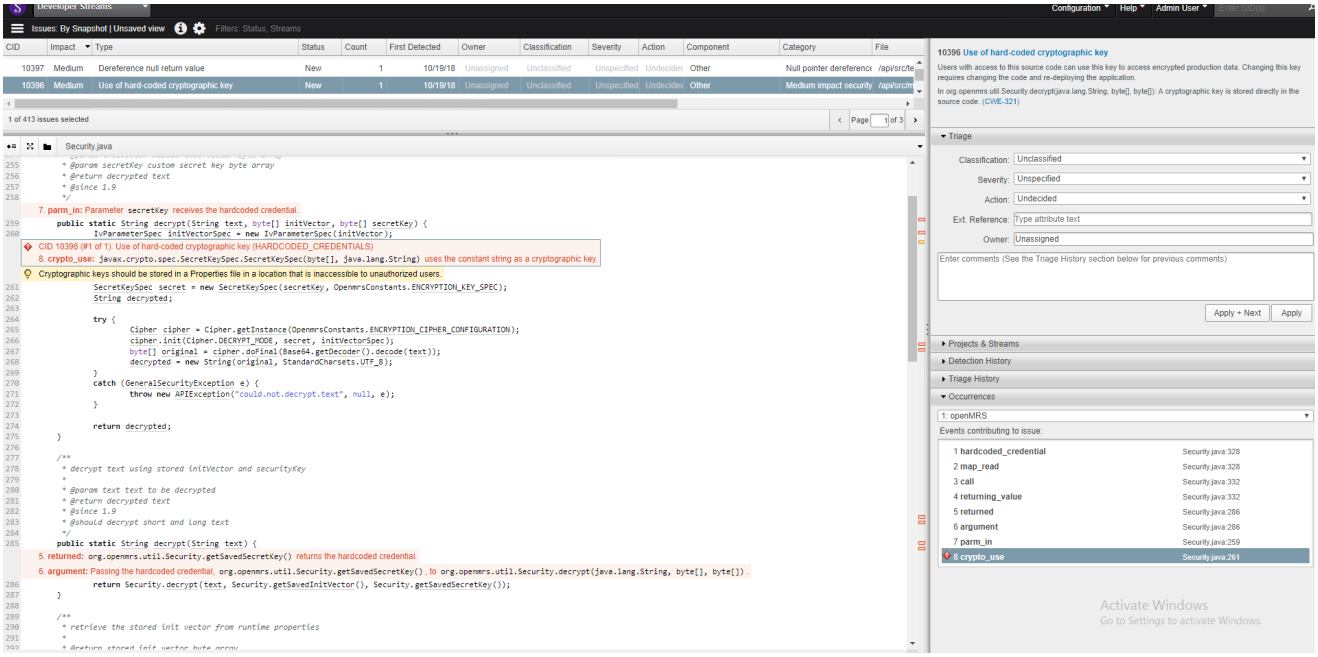
Weakness mitigated by changes:
Runtime exceptions like null pointer exceptions can be handled and dealt with in a graceful manner ,rather than presenting an abrupt exception.

9. Use of hard coded cryptographic key

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

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

Reference to report:



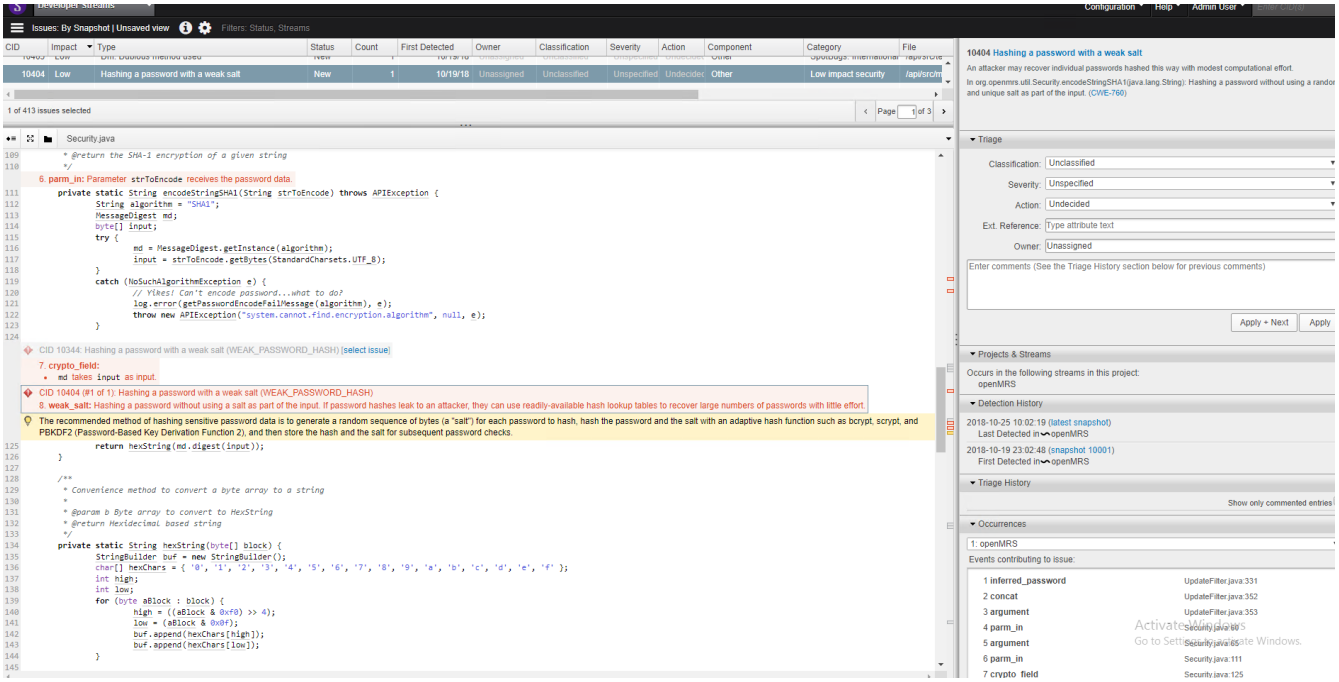
Weakness mitigated by changes:
Suggested changes will help preventing revelation of cryptographic keys to people with access to source code, which can prevent misuse of those keys.

10. Hashing a password with a weak salt

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

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

Reference to report:



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

3. Fuzzing with ZAP:

Test Case ID	Project3_JBroFuzz_1
Test Name	Running JBro Fuzzing tool to identify XSS on create patient
Fuzzer Chosen	XSS: (RuleSet: XSS 101)
Steps to reproduce	1. Setup the ZAP proxy tool as described on this link. ZAP tutorial 2. Use Section to set a breakpoint and ZAP fuzzing from it. 3. Run the Open MRS application on the localhost. 4. Goto the URL: http://localhost:8082/openmrs-standalone/login.htm 5. Login as Admin. Username: admin, password: “Admin123”, select Impatient ward and hit login. 6. Once you are login into OpenMRS. Go to the Register Patient tab in the menu and select it. 7. If you haven’t Open ZAP proxy, select the icon from the desktop and opened it. Till now you would have followed the step 1 which has done setup of proxy. 8. Create a patient with entering details as request on the screen. This can be done by filling first name, Middle Name, Family name & hit enter after entering the Family Name. 9. Now, go to ZAP proxy. And Navigate to openmrs-standalone -> registrationapp -> matchingpatient. There you will see some post request. Select one of the post request and select the Request tab from right hand side. After you are done with this Steps your

	screen will look something similar like Figure 1. 10. Now Select a field value from the Request value pane from the rightmost Down. We have selected **preferred field** value and right click on the selected value. And select the Fuzz from Menu option. 11. Select Payload from Dialog Box and click on Add. 12. From the Drop down of Add Payload select “File Fuzzers” and expand on Jbrofuzz and then look for XSS and expand it also. Select XSS101 form the Ruleset. 13. And click on add button and then OK button and later Start the fuzzer.
Result	Tried out all the probable combination of XSS string and didn’t find any vulnerability.
Vulnerability found	No
Reason for result	Validation is implemented in the OpenMRS source code to handle the XSS attack.
Adjustment to fuzzing rules	Tried other combination of XSS by selecting all the probable XSS string and we see for some of the issue it shows reflected XSS but they return 500 HTTP code.
Mitigation Steps	N/A

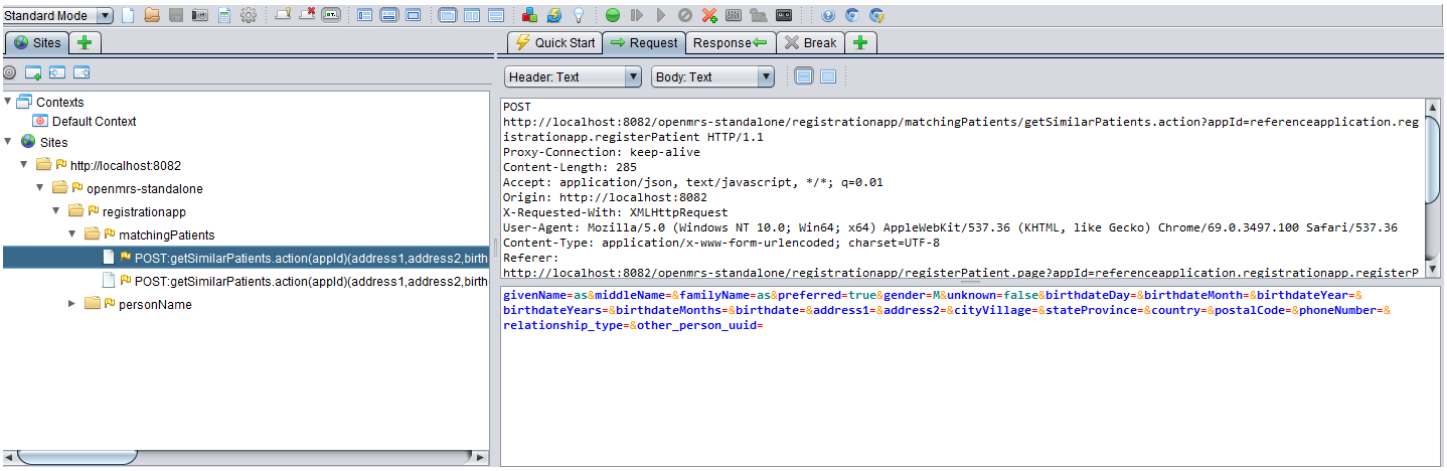


Figure 1: Post request for Jbro fuzz tool

Test Case ID	Project3_JBroFuzz_2
Test Name	Running JBro Fuzzing tool to identify Sql Injection on Create Patient
Fuzzer Chosen	SQL Injection (RuleSet: SQL Injection 101)
Steps to reproduce	1. Setup the ZAP proxy tool as described on this link. ZAP tutorial 2. Use Section to set a breakpoint and ZAP fuzzing from it. 3. Run the Open MRS application on the localhost. 4. Goto the URL: http://localhost:8082/openmrs-standalone/login.htm 5. Login as Admin. Username: admin, password: “Admin123”, select Impatient ward and hit login. 6. Once you are login into OpenMRS. Go to the Register Patient tab in the menu and select it. 7. If you haven’t Open ZAP proxy, select the icon from the desktop and

	opened it. Till now you would have followed the step 1 which has done setup of proxy. 8. Create a patient with entering details as request on the screen. This can be done by filling first name, Middle Name, Family name and hit enter after entering the Family Name. 9. Now, go to ZAP proxy. And Navigate to openmrs-standalone->registrationapp->matchingpatient. There you will see some post request. Select one of the post request and select the Request tab from right hand side. After you are done with this Steps your screen will look something similar like Figure 1. 10. Now Select a field value from the Request value pane from the rightmost Down. We have selected **family Name** value and right click on the selected value. And select the Fuzz from Menu option. 11. Select Payload from Dialog Box and click on Add. 12. From the Drop down of Add Payload select "File Fuzzers" and expand on Jbrofuzz and then look for SQL Injection and expand it also. Select MySQL Injection 101 . OpenMRS uses Mysql db. 13. And click on add button and then OK button and later Start the fuzzer.
Result	Tried all the possible sql attack strings and fuzzer didn't find any vulnerability.
Vulnerability found	No
Reason for result	OpenMRS might be using parameterized query or doing the server side validation of these input.
Adjustment to fuzzing rules	Select all possible Sql injection string from the list of fuzzer and try to run the fuzzer. Fuzzer don't show us any sign of vulnerability on this.
Mitigation Steps	N/A

Test Case ID	**Project3_JBroFuzz_3**
Test Name	Running JBro Fuzzing tool to identify Buffer overflow on Edit Patient
Fuzzer Chosen	Buffer Overflow (RuleSet : Long Strings of aaa's)
Steps to reproduce	1. Setup the ZAP proxy tool as described on this link. ZAP tutorial and also open the ZAP tool and configure local proxy as needed. 2. Now in that doc, follow the steps listed under section to set a breakpoint and ZAP fuzzing from it. 3. Run the Open MRS application on the localhost. 4. Goto the URL: http://localhost:8082/openmrs-standalone/login.htm 5. Login as Admin. Username: admin, password: "Admin123", select Isolation ward and hit login. 6. Once you are login into OpenMRS. Go to the Find Patient Record tab in the menu and then on new screen,enter John in the search box and hit enter. 7. Now in the search results that appears,select the row with record of John Smith and click on it. 8. Now patient details page appears,then click on Edit button.

	9. Now on the page that appears,change some details there and click on Save Form and then click on Confirm button. 10. Go to ZAP and locate POST request for it, which in this case is POST:editSection 11. Now go to the Request tab,and highlight the familyname value there and right click to select and open Fuzz. 12. Now click on Payloads button and then click on Add button. 13. Now select File Fuzzers from the Type dropdown and then expand the jbrofuzz and then select Buffer overflow checkbox listed under it and then click on Add button and then click ok and then click on Start Fuzzer.
Result	Tried all the accepted patterns,but didn't found out vulnerabilities.
Vulnerability found	No
Reason for result	Validations had been implemented in openmrs for checking maximum field length.
Adjustment to fuzzing rules	Will try the long list of numbers also with fields of numerical value.
Mitigation Steps	N/A

Test Case ID	**Project3_JBroFuzz_4**
Test Name	Running JBro Fuzzing tool to identify Injection on Edit Patient
Fuzzer Chosen	Injection(RuleSet : MySQL Injection 101)
Steps to reproduce	14. Setup the ZAP proxy tool as described on this link. ZAP tutorial and also open the ZAP tool and configure local proxy as needed. 15. Now in that doc, follow the steps listed under section to set a breakpoint and ZAP fuzzing from it. 16. Run the Open MRS application on the localhost. 17. Goto the URL: http://localhost:8082/openmrs-standalone/login.htm 18. Login as Admin. Username: admin, password: "Admin123", select Isolation ward and hit login. 19. Once you are login into OpenMRS. Go to the Find/Create Patient page at http://localhost:8082/openmrs-standalone/findPatient.htm.. 20. Now in the Find Patient section enter name as 'John' and click enter. 21. Go to ZAP and locate POST request for it, which in this case is POST:DWRPatientService.findCountAndPatients.dwr(callCount) 22. Now go to the Request tab,and highlight the value John stored in c0-param0=string variable,then right click to select and open Fuzz. 23. Now click on Payloads button and then click on Add button. 24. Now select File Fuzzers from the Type dropdown and then expand the jbrofuzz and then select Injection checkbox listed under it and then select MySQL Injection 101 ,and click on Add button and then click ok and then click on Start Fuzzer.
Result	No vulnerabilities found, for all the patterns listed in ruleset of MySQL injection 101

Vulnerability found	No
Reason for result	System throws exception of allowscripttagRemoting as false
Adjustment to fuzzing rules	Tried adding more injection patterns
Mitigation Steps	N/A

4. Client-side bypassing with ZAP:

1. Start ZAP from the shortcut on the desktop and once the application is open, change the port the proxy server is on by going to Tools -> Options in the menu bar and select “Local Proxy." Change the port to “8008” or a port of your choice.
2. In the Google Chrome browser inside the VCL image, click the 3 dot in the top right corner (Chrome Options) and go to Settings -> Show Advanced Settings -> Open proxy settings
3. In the new window that appears, under the Connections tab, click “LAN Settings”. Make sure the “Use a proxy server for your LAN" box is selected, and enter the following information: Address: localhost Port: 8008
4. After entering this information, click Advanced" and make sure the “Use the same proxy server for all protocols" box is checked. Click OK on all internet settings boxes and close the Chrome settings page.
5. Start OpenMRS, by clicking on the ‘Launch OpenMRS’ icon on the VCL desktop.Make sure your openMrs is running on 8082.
6. After OpenMRS starts, navigate to the OpenMRS instance at: “http://localhost:8082/openmrs-standalone “ in the Chrome browser.This should now pop up in the “Sites” list in ZAP.

Test Case ID	**Project3_Bypass_1**
Test Name	SQL Injection on Login page of OpenMRS using ZAP proxy
Steps to reproduce	1. After setting the ZAP proxy following the above steps. Open the Login page of Open MRS application in Chrome browser. URL: http://localhost:8082/openmrs-standalone/login.htm 2. Try to do a successful login attempt using admin. Username: admin, Password: “Admin123”, select Impatient ward. 3. Go to the ZAP proxy and in sites section navigate to “http://localhost:8082” -> openmrs-standalone, in this you will see a Post request for login page (Post: login.htm...). 4. Right click on it and Select breakpoint and Click OK. Breakpoint on login page will successfully set. 5. Logout from the openmrs application current session. 6. Go to ZAP Proxy and create a new Session. 7. Once the session is created successfully then go to the login page of openmrs. And Enter admin username and some password and select

	impatien ward. 8. Now in ZAP proxy modify the login Post request **Original request:** username=admin&password=123&sessionLocation=6&redirectUrl=%2Fopenmrs-standalone%2Freferenceapplication%2Fhome.page **Change Request:** username=admin&password=' or '1'='1&sessionLocation=6&redirectUrl=%2Fopenmrs-standalone%2Freferenceapplication%2Fhome.page 9. Once you have modified the request send this request it using Play button from the menu tab of ZAP Proxy tool. If required Submit the request again until there is no more action required in ZAP to clear the request. 10. In Chrome browser you will be redirected to the Login page of the application again and see an error message "**Invalid username/password. Please try again.**"
Description	We have modified the client request for login into OpenMRS and tried the sql injection on it. SQL Injection was failed. And suitable error message is displayed on the application.
Input Field	password
Initial User Input	"123"
Malicious Input	' or '1'='1
Expected result	Login attempt failed with suitable error Message.
Actual Result	Login attempt failed with error message "**Invalid username/password. Please try again.**"
Test Result	Pass

Test Case ID	**Project3_Bypass_2**
Test Name	XSS on Register Patient record in OpenMRS using ZAP proxy
Steps to reproduce	1. After setting the ZAP proxy following the above steps. Open the Login page of Open MRS application in Chrome browser. URL: http://localhost:8082/openmrs-standalone/login.htm 2. Login to Openmrs using Username: admin, Password: "Admin123", select Impatient ward. And hit login. 3. From home page select the register patient. 4. Go to ZAP proxy and start a new Session. [If you have previous breakpoint on login page remove it from ZAP proxy app in lower pane "Breakpoint". Deselect all the breakpoints.] 5. Go to Chrome and fill the details for a new Patient. You can select your own value, but I have selected the values: Given Name: "arpit", "Family Name": "kumar", Gender: Male, BirthDate: 01 - Nov- 2000 . address: Raleigh, Phone Number: "9191234". After filling the relative don't Confirm and go to ZAP proxy. 6. Go to the ZAP proxy and in sites section navigate to "http://localhost:8082" -> openmrs-standalone -> registration app->registerpatien, in this you will see a

	Post request for Patient page (Post:submit.action..). 7. Right click on it and Select breakpoint and Click OK. Breakpoint on registration page will successfully set. 8. Logout from the openmrs application current session. 9. Go to ZAP Proxy and create a new Session. 10. Once the session is created successfully then go to the login page of openmrs. And Enter admin username and some password and select impatien ward. After successful attempt go to register patient and add some details for new patient and confirm the details. 11. Once you confirm browser hangs and ZAP will become active window. 12. Go to ZAP proxy and go to the location mentioned in above steps 6 and modify the request as below. 13. Now in ZAP proxy modify the register patient Post request **Original request:** givenName=arpit&middleName=&familyName=kumar&preferred=true&gender=M&unknown=false&birthdateDay=01&birthdateMonth=11&birthdateYear=2000&birthdate=2000-11-01&address1=raleigh&address2=&cityVillage=&stateProvince=&country=&postalCode=&phoneNumber=91912345&relationship_type=&other_person_uuid= **Change Request:** givenName=arpit&middleName=&familyName=<script>alert("hello")</script>&preferred=true&gender=M&unknown=false&birthdateDay=01&birthdateMonth=11&birthdateYear=2000&birthdate=2000-11-01&address1=raleigh&address2=&cityVillage=&stateProvince=&country=&postalCode=&phoneNumber=91912345&relationship_type=&other_person_uuid= 14. Once you have modified the request send this request it using Play button from the menu tab of ZAP Proxy tool. If required Submit the request again until there is no more action required in ZAP to clear the request. 15. In Chrome browser you will be redirected to the register page and you see all the details. Even though the name is our script, XSS is not reflected on this page.
Description	We have modified the register patient request using ZAP proxy. We used family name as field to modify because this info is reflected after registering the page.
Input Field	Family name
Initial User Input	Family name: “kumar”
Malicious Input	<script>alert(“hello”)</script>
Expected result	XSS reflection fails and patient is registered with updated value.
Actual Result	XSS does not reflects and new patient is created and redirect to patient page.
Test Result	Pass

Test Case ID	**Project3_Bypass_3**

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

Actual Result	XSS reflection fails and patient details get updated and redirect to patient page.
Test Result	Pass

Test Case ID	**Project3_Bypass_4**
Test Name	Intercept and Change Admin password in OpenMRS using ZAP proxy
Steps to reproduce	1. After setting the ZAP proxy following the above steps. Open the Login page of Open MRS application in Chrome browser. URL: http://localhost:8082/openmrs-standalone/login.htm 2. Login to Openmrs using Username: admin, Password: “Admin123”, select Impatient ward. And hit login. 3. Now go to url http://localhost:8082/openmrs-standalone/admin/users/users.list 4. Go to ZAP proxy and start a new Session. [If you have previous breakpoint on login page or register page remove it from ZAP proxy app in lower pane “Breakpoint”. Deselect all the breakpoints.] 5. Search for “ admin” by typing it in find user on name textbox and then click on Search button. 6. Now click on the admin hyperlink that appears in search results and then there type password new password in user’s password and in confirm password and then click on Save user. 7. Go to the ZAP proxy and in sites section navigate to “http://localhost:8082” -> openmrs-standalone -> admin->users, in this you will see a Post request for users form (Post:users.form..). 8. Right click on it and Select breakpoint and Click OK. Breakpoint on user password change page will be successfully set. 9. Logout from the openmrs application current session. 10. Go to ZAP Proxy and create a new Session. 11. Once the session is created successfully then go to the login page of openmrs. And then go to http://localhost:8082/openmrs-standalone/admin/users/users.list ,search for admin and then change password as mentioned in the earlier steps and click on Save user.Once you confirm browser hangs and ZAP will become active window. 12. Go to ZAP proxy and go to the location mentioned in above steps 7 and modify the request as below. 13. Now in ZAP proxy modify the users form Post request a. **Original request:** b. userId=1&person_id=1&person.names%5B0%5D.givenName=Super&person.names%5B0%5D.middleName=&person.names%5B0%5D.familyName=User&person.gender=M&username=&**userFormPassword=Admin1234&confirm=Admin1234**&roleStrings=Provider&roleStrings=System+Developer&secretQuestion=&secretAnswer=&property=loginAttempts&value=0&property=lockoutTimestamp&value=&action=Save+User **c. Change Request:** d. userId=1&person_id=1&person.names%5B0%5D.givenName=Super&person.names%5B0%5D.middleName=&person.names%5B0%5D.familyName=User&person.gender=M&username=&**userFormPassw**

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

Test Case ID	**Project3_Bypass_5**
Test Name	Intercept and Change PatientSearch queries in OpenMRS using ZAP proxy
Steps to reproduce	1. After setting the ZAP proxy following the above steps. Open the Login page of Open MRS application in Chrome browser. URL: http://localhost:8082/openmrs-standalone/login.htm 2. Login to Openmrs using Username: admin, Password: “Admin123”, select Impatient ward. And hit login. 3. Now go to url http://localhost:8082/openmrs-standalone/admin/patients/index.htm and there select Manage patients tab. 4. Go to ZAP proxy and start a new Session. [If you have previous breakpoint on login page or register page remove it from ZAP proxy app in lower pane “Breakpoint”. Deselect all the breakpoints.] 5. Type in “John” by typing it in find user textbox and then click on Search button. 6. Go to the ZAP proxy and in sites section navigate to “http://localhost:8082” -> openmrs-standalone -> ms>call->plaincall, in this you will see a Post request for findPatients (Post:DWRPatientService.findCountAndPatientsWithVoided.dwr)

	7. Right click on it and Select breakpoint and Click OK. Breakpoint on user password change page will be successfully set. 8. Now again go to http://localhost:8082/openmrs-standalone/admin/patients/index.htm ,search for John and click enter,now browser hangs and ZAP will become active window. 9. Go to ZAP proxy and go to the location mentioned in above steps 7 and modify the request as below. 10. Now in ZAP proxy modify the findCount Post request **a. Original request:** callCount=1 page=/openmrs-standalone/admin/patients/index.htm httpSessionId= scriptSessionId=0E79A3185C8E2E3EC4DB87C0A06B1425528 c0-scriptName=DWRPatientService c0-methodName=findCountAndPatientsWithVoided c0-id=0 c0-param0=string:John c0-param1=number:0 c0-param2=number:10 c0-param3=boolean:true c0-param4=boolean:false batchId=2 **b. Change Request:** callCount=1 page=/openmrs-standalone/admin/patients/index.htm httpSessionId= scriptSessionId=0E79A3185C8E2E3EC4DB87C0A06B1425528 c0-scriptName=DWRPatientService c0-methodName=findCountAndPatientsWithVoided c0-id=0 c0-param0=string:Betty c0-param1=number:0 c0-param2=number:10 c0-param3=boolean:true c0-param4=boolean:false batchId=2 11. Once you have modified the request send this request it using Play button from the menu tab of ZAP Proxy tool. If required Submit the request again until there is no more action required in ZAP to clear the request. 12. In Chrome browser you will be redirected to the search results pane,now you can see we have search results updated for name Betty provided with ZAP.
Description	We have modified the c0-param0=string value for getting search results for a person name.
Input field	c0-param0
Initial User Input	c0-param0=string : John
Malicious Input	c0-param0=string: Betty
Expected result	Search query should not have been intercepted,changed and reflected in search

	results.
Actual Result	Search query successfully intercepted,changed and reflected in search results.
Test Result	Fail

5. Security Requirements:

Requirement ID	Security Requirements
OpenMRS_SR1	Sufficient logging should be done so that it's possible to identify who accessed patients' records or created a new patient record.
OpenMRS_SR2	In case of errors or failures, limited amount of error information should be displayed to the user, which prevents leaking of any software version, exception type, stack trace.
OpenMRS_SR3	User privileges based access controls should be established so that the users with only authorized access are able to find/create patient records.
OpenMRS_SR4	Location based access controls should be established so that it can manage the access to all services based on location. For example, user who has multiple locations access (admin) should be able to view patients in all locations.
OpenMRS_SR5	Input fields must be properly validated on client and server side to prevent SQL Injection and Cross Site Scripting attacks. We should also constrain input, sanitize input, reject known bad input for the fields using for searching patient's records, and creating new patient record.
OpenMRS_SR6	Every request made to find/create patient records must use HTTPS protocol through SSL/TLS certificates which allows an encrypted secure connection between client and server hence safeguarding from man-in-the-middle attack.
OpenMRS_SR7	Strong access control measures should be implemented so that patients records must be kept confidential and not disclosed to an unauthorized party which protects from sensitive data exposure vulnerability/attack.
OpenMRS_SR8	Users passwords should be protected by salted password hashing while registering user or in login module, safeguarding from dictionary and brute force attacks, lookup tables, reverse lookup tables, rainbow tables. This is for the login/register module.
OpenMRS_SR9	Data protection is required which means platform security configuration, use of encryption at rest and in transit and entire life cycle secure is needed. This requirement is for the entire system.
OpenMRS_SR10	Audit policies should be detailed enough to be useful. This is for the entire system.
OpenMRS_SR11	Data consistency and correctness of health care information in the database are key requirements. Incorrect data can have bad consequences, such as incorrectly diagnosing a patient. This is for the database system.

OpenMRS_SR12	Privacy of protected health information should be implemented as per HIPAA Privacy Rule which includes individual's right to adequate notice of uses and disclosures of his or her protected health information that may be made by the covered entity and the individual's rights and the covered entity's duties with respect to the protected health information. This requirement is for the entire system.

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