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

Group Members

Rushi Bhatt (rbhatt)

Neetish Pathak (npathak2)

Bhavya Bansal (bbansal)

Aswin Anil Kumar (aanilku)

For Part 3, you should use your local copy of OpenMRS, the code for OpenMRS, and the class VCL image of OpenMRS, as appropriate.  
  
**0. Module Selection**For each of the below activities, pick a single module of OpenMRS to diagram and attack. This could be the 'Find/Create Patient' module, the 'Add New Concept in Dictionary', 'Add/Edit User', et cetera. If you're not sure if your choice is appropriate, ask on Piazza.

Our primary module for the tests is Add/Edit/Delete user as used in previous parts of this project.

However, we used the add/view patient as the additional modules for the Audit/logging implementation since it had more relevant use cases to encounter sensitive medical data.

We also used login module as the additional module to capture security requirements in problem 5 of this project. That’s because login module provides a better scope and more use cases relevant to security requirements.   
  
**1. Audit/logging implementation (20 points)**  
Write 10 test cases that will add/edit/delete/view sensitive medical data. In the expected results column, write what you expect to be logged when you run the test case. Find OpenMRS's transaction log. Write what is logged in the actual results column. Comment on the adequacy of OpenMRS's logging.

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

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

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

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

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

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

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

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

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

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

**2. Static analysis with Fortify (20 points)**   
Find the instructions for getting Fortify going on OpenMRS here.   
Review the security reports for "openmrs-api". Based upon these reports, create a prioritized list of **ten** changes the systems developers should make to OpenMRS to correct the deficiencies found. For each change, document the change required, what weakness the change mitigates, and provide a cross-reference back to the originating report(s) where the issue was documented. The originating report is the "Static Analysis Results" within the Fortify Perspective. These results can be exported to PDF through using the "Generate Legacy Report" under the Fortify menu.

Based on Static analysis using fortify

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

|  |  |
| --- | --- |
| OpenMRS Deficiency 2 | **Server Side Template Injection:** The call to evaluate() in VelocityMessagePreparator.java on line 60 evaluates user-controlled data as a template engine's template, allowing attackers to access the template context and in some cases inject and run arbitrary code in the application server.    engine.evaluate(context, writer, "template", template.getTemplate()); |
| Required Change | * Do not allow users to provide templates. * If user-provided templates are necessary, perform careful input validation to prevent malicious code from being injected in the template. Exp, use Whitelisting, blacklisting and input sanitization to control the template inputs. |
| Change mitigates | * Input constraints on user provided templates can prevent attackers to inject expressions that will expose context data or run arbitrary commands on the server. |
| Cross-reference | vuln2.png |

|  |  |
| --- | --- |
| OpenMRS Deficiency 3 | **Command Injection:**The method execMysqlCmd() in MigrateDataSet.java calls exec() with a command built from untrusted data. This call can cause the program to execute malicious commands on behalf of an attacker. The attacker explicitly/implicitly controls what the command is.    Process p = (wd != null) ? Runtime.getRuntime().exec(cmds, null, wd) : Runtime.getRuntime().exec(cmds); |
| Required Change | * Do not allow users to have direct control over the commands executed by the program * In cases where user input must affect the command to be run, use the input only to make a selection from a predetermined set of safe commands * Blacklist/Whitelist the input parameters to the command execution function. Use fortify,Spring MVC,Strut and other static analysis frameworks to identify and perform input validation. * Commands should be controlled by the application and executed using an absolute path * Command values and paths read from configuration files or the environment should be sanity-checked against a set of invariants that define valid values. * Use principle of least privilege when executing any external commands |
| Change mitigates | * Blacklisting selectively rejects or escapes potentially dangerous characters before using the input * Whitelist can effectively accept only the input composed exclusively of characters in the approved set. * Sanity checks against the path and values can enforce the safety of the environment and prevent attacker to exploit the command execution. * Giving least privileges to external command can minimize the damage. |
| Cross-reference | vuln3.png |

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

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

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

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

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

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

|  |  |
| --- | --- |
| OpenMRS Deficiency 10 | **Server-Side Request Forgery:** The function openConnection() on line 720 initiates a network connection to a third-party system using user-controlled data for resource URI. An attacker may leverage this vulnerability to send a request on behalf of the application server since the request will originate from the application server internal IP.    String loc = http.getHeaderField("Location");  ……...  c = target.openConnection(); |
| Required Change | * Do not establish network connections based on user-controlled data * Use a level of indirection: create a list of legitimate resource names that a user is allowed to specify, and only allow the user to select from the list. * As an added precaution, blacklist/whitelist the URL parameters to the target location on which you want to open the connection. |
| Change mitigates | * With indirection approach the input provided by the user is never used directly to specify the resource name, thereby preventing any kind of input based forgery attack * Blacklisting selectively rejects or escapes potentially dangerous characters before using the input * Whitelist can effectively accept only the input composed exclusively of characters in the approved set. |
| Cross-reference | vuln10.png |

**3. Fuzzing with ZAP (15 points)**  
Use the jbrofuzz rulesets (introduced in the initial ZAP activity) to perform a fuzzing exercise on OpenMRS with the following vulnerability types: Injection, Buffer Overflow, XSS, and SQL Injection.   
Pick at least one ruleset for each type of vulnerability listed; you don't need to run all of them. The ruleset should be appropriate for the target field and backend (for example, you wouldn't run an MSSQL ruleset against an application running mySQL).   
 Each fuzzing exercise should be described as a test case, in the same format as part 1. Provide a unique identifier for each test.  
Report the fuzzers you chose for each vulnerability type along with the results, and what you believe the team would need to do to fix any vulnerabilities you find. If you don't find any vulnerabilities, provide your reasoning as to why that was the case, and describe how you would adjust the fuzzing rules you used.  
The fuzzing should be performed on the VCL Class Image ("CSC 515 2017 Windows 10 Class Image"). Use the standalone version of OpenMRS installed (version 2.6.1).

We have tested the fuzzing activity on the Add user module and fuzzing the username field

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

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

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

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

**4. Client-side bypassing with ZAP (15 points)**  
Plan 5 test cases in which you stop user input in OpenMRS with ZAP and change the input string to an attack.   
Document the page URL, the input field, the initial user input, and the malicious input. Describe what "filler" information is used for the rest of the fields on the page (if necessary).   
Run the test case and document the results. Provide a unique identifier for each test in the same style as part 1.  
This testing should be performed against the standalone version of OpenMRS (version 2.6.1) installed on the class image.

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

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

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

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

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

**5. Security Requirements (20 points)**   
Develop 10 security requirements for OpenMRS. Focus on your chosen module; if you run out of requirements for your modules, you may split your requirements over multiple modules. Each requirement should have a unique identifier. Use the class notes as a guide for structuring your requirements.  
2 points per requirement.

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

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

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

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

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

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

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

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

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

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