OpenMRS HIPAA Assessment

July 4, 2016

Dr. Steven P. Crain (OpenMRS ID scrain777)
Department of Computer Science
State University of New York, Plattsburgh

Contributing Students (partial list):
Trevan A.
Sergio A.
Joe A.
Marisa A.
Nicholas B.
Alex C.
Pat C.
Zachary D.
Mouctar Diallo (OpenMRS ID mdiald)
Xiaocan D.
Dan E.
Jared F.
Kevin G.
Marie H.
Kunal K.
Gabrielle L.
Robert L.
Leonardo Alves Miguel (OpenMRS ID lmigu001)
Lucas Narciso (OpenMRS ID lucasnar)
Richard Pinter (OpenMRS ID rpint001)
Seung Ho R.
Akshay S.
Brigham T.
Alex Torres (OpenMRS ID 10LeftFeet)
Nhat V.
Kyle Williams (OpenMRS ID region39)

Table of Contents

Executive Summary

Students at Plattsburgh State University of New York conducted security assessments of OpenMRS during May 2015 and May 2016. Based on the assessments, we recommend specific improvements related to authentication, auditing and documentation.

Our assessment was based on HIPAA regulations, which provide important guidance with respect to the security of health information. HIPAA regulations would be directly applicable if OpenMRS were used in the United States. The students assessed the OpenMRS Core API, database and reference application with attention to access control, accountability and privacy. This assessment followed a well-documented procedure, and can be used as a baseline for regular repeated assessments.

One way that security professionals assess security is using the "CIA triad." We found that OpenMRS is weak at protecting *confidentiality*, because, once users log in, they can access any protected health information without restriction and without leaving an audit trail. On the other hand, OpenMRS ensures that only authorized users make changes in the data and generally keeps track of significant changes (*integrity*). The students did not assess *availability*, as that is not a major concern in the HIPAA regulations.

Another way that security professionals organize assessment is using the "AAA triad." OpenMRS has solid *authentication* and *authorization* support, but has inadequate support for *accountability*.

We also measured OpenMRS against commonly used design principles. We found that the overall architecture of OpenMRS was very conducive to security and that OpenMRS has done well at balancing security and usability. However, OpenMRS has some major security concerns. The default admin password violates the principle of fail-safe defaults. Also, there is no support for separating administrative privileges to different users.

From our findings, we determined the most important security changes for OpenMRS. Most importantly, the default admin password should be corrected by implementing a password expiration or password quality policy. Additionally, OpenMRS should implement session timeouts and account locking with repeated login failures. Although more difficult to implement, we provided some feature recommendations for a much-needed auditing system. Further, we recommended a section providing security guidance in the implementers' documentation. Finally, changes should be made so that administrative responsibilities can be divided across multiple administrators. We believe that these recommendations will make a meaningful difference in the security of OpenMRS implementations.

Background

This assessment focuses on how well OpenMRS complies with regulations based on the U.S. Health Information Protection and Accountability Act (HIPAA). Although these regulations are not legally relevant in most of the world, they provide sound guidance to many of the security aspects of handling medical information. Improving OpenMRS's compliance with HIPAA regulations would enhance confidence in its security and open the possibility of deployments in the U.S.

HIPAA does not make any specific requirements for what security is required. Instead, it primarily gives patients control over whether providers share information with other organizations and requires the ability to report to patients if data is shared without their permission. As a result, the main focus of HIPAA is on restrictions to read-access to data, and keeping records of who accesses data and how it is used. The other important aspects of security (protecting information against being changed improperly and protecting the availability of the system) are not assessed in this report, although they are clearly critical for overall safety of an OpenMRS installation.

Students in an upper-level computer security course at SUNY Plattsburgh provided the content of this assessment. The students were provided training on the requirements of HIPAA and the process of conducting a security assessment. The assessment was divided into 9 focus areas, with different groups

of students responsible for conducting the assessments in each area. The assessments vary in quality, depending largely on the effort the students invested in the assignment. However, the instructor has 12 years of industry experience with operating system, network, application and database security and has worked with the students to distill the recommendations into a form that we hope will provide useful guidance for improving the security of OpenMRS and a useful model for future assessments.

Methodology

The nine focus areas can be grouped according to three OpenMRS components assessed on three aspects of security. Some students focused on the reference application, which is used by most OpenMRS implementers and provides a security model that other applications will likely follow. Other students focused on the core API, which provides security strengths and weaknesses to all OpenMRS implementations. The remaining students examined security of the database that stores the OpenMRS data.

Students were further divided by the aspect of security that they assessed. Computer security professionals talk about security using two ``triads.'' The first triad, "CIA,'' relates to the three goals of security: confidentiality, integrity and availability.

- Confidentiality means that information can only be accessed with proper authorization and used in appropriate ways.
- Integrity means that the information in a system is changed only following established procedures, so that the information remains correct.
- Availability means that the system is protected so that it is available for legitimate uses at all times.

The second triad, "AAA," provides three best-practices that form the foundation of strong security. These are authentication, authorization and accountability.

- Authentication requires users to prove who they are before they can access a system.
- Authorization puts restrictions on what users can do in a system.
- Accountability keeps track of what users do in the system so that it is possible to hold individuals accountable for their actions.

The primary concerns of HIPAA are confidentiality and accountability, so we had groups of students focused on each of these aspects of security. The remaining group of students examined the authentication and authorization systems.

Each group of students followed a security assessment consisting of the following steps. Details of their instructions are available at http://foss2serve.org/index.php/OpenMRS_Security_Assessment.

1. Security Context. The students were asked to explore what OpenMRS is and how it is used. Every situation is unique, and the security assessment needs to take into account the unique challenges of a medical record system and the security needs of its users and developers.

2. Identify assets. Assets are resources of value, whether they are valuable to the owners or to other people. There are many types of assets, from information to computers to employees. Because of their value, assets are generally the targets of attacks, and so the students were asked to identify assets and consider the possible attacks against each asset.
3. Identify threats. The students next assessed the threats to those assets. Threats are any situation that could happen that would impair the confidentiality, integrity or availability of any asset. Each threat is classified according to how likely it is to happen, and the severity of the consequences if it is. Together, the likelihood and severity give an estimate of the risk or average cost of each threat.
4. Evaluate controls. The students examined OpenMRS and its documentation looking for any measures that have been taken to target the specific threats. They then proposed additional measures that should be considered if the existing measures seemed inadequate.
5. Evaluate security principles. The students assessed OpenMRS based on how well it adhered to established security principles.
6. Recommendations. The students then summarized what they found and their recommendations.

Security Context

OpenMRS is used extensively in clinical settings throughout the world. The broad range of settings introduces unique challenges in terms of possible threats. Moreover, the OpenMRS implementers have the expectation that OpenMRS provides appropriate security. If OpenMRS were expanded into clinical use in the United State, there would be additional considerations for HIPAA compliance.

OpenMRS is used in many clinics, especially throughout India and Africa. This means that OpenMRS implementers face many threats that would be unfamiliar to a medical facility in the United States. Political corruption and bribery are rampant in many of the areas where OpenMRS is used, which increase the risk of clinical employees altering or stealing data for political purposes. On the other hand, there is a smaller risk of theft of large amounts of data and insurance fraud, which are unique aspects of the threat landscape in the United States.

OpenMRS is popular in resource constrained environments, where the implementers do not have the expertise needed for a local risk assessment. Instead, they expect the OpenMRS developers to have created a product that provides what they need in security. Unfortunately, the OpenMRS development team also lacks the security expertise needed to meet this expectation. We hope that this assessment will provide valuable direction so that OpenMRS will be more secure in a wide variety of environments.

Use of OpenMRS in the United States poses its own security challenges. For clinical deployment in the U.S., OpenMRS would need to have stronger controls protecting access to protected health information. Installations in the US would also face unique threats based on the value of protected health information to identity thieves and clinicians manipulating data to commit insurance fraud or to cover up evidence of malpractice. In the U.S., then, accountability is a particularly important aspect of security in medical records systems.

Summary of Findings

General

There are no authentication timeouts. If a clinician leaves a computer unattended, an intruder could easily access the system. This would allow the intruder to access or modify confidential records with the same authorizations as the clinician.

HIPAA regulations place a great emphasis on establishing procedures that, when followed, ensure HIPAA compliance. To meet this requirement, the medical record system must have be tied to the established procedures, either through having the medical record system enforce the procedures or by having an audit trail that can be correlated to other documentation of the procedures. OpenMRS cannot currently support either of these options.

Confidentiality

Confidentiality refers to protecting information from inappropriate disclosure. In general, we found that OpenMRS is weak at protecting confidentiality. Once a user is authenticated, OpenMRS places very few restrictions on what the user can see.

Integrity

Integrity refers to protecting information from inappropriate modification. In general, we found that OpenMRS is moderate at protecting integrity. OpenMRS has a solid role-based access control system that prevents unauthorized modifications.

Availability

Availability refers to ensuring that OpenMRS is ready to be used for authorized purposes when needed. Availability is critical for a health information system, since the failure of the system could result in needless death of patients. However, availability falls outside of the scope of this assessment because we are focusing on compliance with HIPAA regulations. That said, we did note while doing our assessment that OpenMRS generally has solid availability, although we did not encounter mechanisms to measure the availability of OpenMRS or to take corrective actions if it fails.

Authentication

Authentication refers to establishing the identity of a user who is trying to interact with OpenMRS. Authentication in the API and application are based on three factors: knowledge of the username and password and network access to the API or application. This level of authentication is typical in clinical settings, regardless of the MRS in use. However, there are no authentication timeouts, so if a clinician leaves a computer unattended, anyone can interact with the system.

Sadly, none of the students thought to investigate what happens if a user installs an unauthorized application parallel to the reference application. My guess, based on what I know of OpenMRS architecture, is that a rogue application would be able to access API services just like the authorized application. This would not bypass authentication, but would bypass any business processes enforced in the user interface application. This can be easily prevented by configuring Tomcat (or whatever Java

container is being used) to restrict access to the API by IP address, but we doubt that very many OpenMRS installations have made that configuration change.

The API accesses the database on behalf of the users, using a single username (openmrs_user) and password (randomly selected) established for this purpose. The installation process we followed also created an openmrs_auditor database login, but we did not note this login being used for anything. It is possible that the API uses this second username and password when it is in the accounting logic. The openmrs_user login has full access to the openmrs database. This is a violation of standard practice, which uses separate usernames for managing database schema, performing maintenance operations and access on behalf of users.

Authorization

Authorization refers to determining whether an individual is allowed to do what she is requesting to do. OpenMRS uses a role-based authorization model, where the administrator can define roles, configure in detail what the role is authorized to do and manage the roles assigned to users. OpenMRS comes with many pre-established roles that are appropriate in a wide range of clinical settings. This makes it very easy for implementers to create accounts for their users that have only the permissions necessary for their job responsibilities.

Accountability

Accountability refers to keeping track of interactions with OpenMRS is a way that individuals can be kept accountable for their actions. Accountability is a critical part of a medical record system, protecting against fraud, corruption, and human error and helping detect intrusion. We found very little support for accountability in any part of OpenMRS. The exception is that many data modifications are non-destructive, meaning that the old version of the record remains in the database, and the identity of the person making the change becomes a permanent record in the database. This is an excellent design and has many elements of an auditing system, but is inadequate for ensuring accountability.

Design Principles

Security professionals use a number of standard design principles to help ensure the security of a system. The design principles cannot be blindly applied, and they may be more or less relevant to a particular project, but, taken as a whole, they provide valuable design guidance for OpenMRS.

Economy of Mechanism: Grade B

Economy of mechanism refers to keeping security simple. OpenMRS uses Java aspects to configure the type of authorization needed for each method. This style of coding the authorization rules makes it easy to manage the requirements of each method, but it makes it hard to determine where the code is that enforces the requirements. In balance, it seems a reasonable choice. Overall OpenMRS does well at keeping security simple in spite of its large size and complexity.

Fail-safe Defaults: Grade D

Fail-safe defaults refers to ensuring that the system, as installed, will be secure even if the implementers fail to complete some of the recommended configuration steps. The general idea is to make the system

as secure as it can be when it first installs, and require the implementers to intentionally shut off any security components that are overkill for their risk environment. OpenMRS has made improvements on fail-safe defaults over the past year, but still installs with a fixed administrator username and password that has unrestricted access to the system.

Complete Mediation: Grade A-

Complete mediation refers to restricting access to the system so that the only way in is through the security mechanisms. OpenMRS uses an API layer that is intended to provide complete mediation, which is an excellent start. However, it is still possible, and even required for maintenance, to access the database directly, either connecting to the database engine or accessing the data files.

Separation of Privilege: Grade E

Separation of privilege refers to the practice of splitting permissions up amongst multiple people, so that actions of one rogue or hacked user account can cause a limited amount of damage. In the database, OpenMRS uses a single account with unrestricted access for everything that it does. Moreover, the only kind of administrator that OpenMRS supports is one with unrestricted power in the system.

Separation of privileges is closely related to the concept of segmenting assets so that an attacker who gains access to some of the assets can only get a small fraction of them. OpenMRS does not provide any way to divide the medical records into smaller sets, either logically (restrict access to patients based on their provider relationships) or physically (place data into different databases protected by different passwords).

Least Privilege: Grade C

Least privilege refers to the practice of giving each user or system the minimum permissions needed for their job responsibilities. The role-based authentication system makes it easy for administrators to achieve least privilege at a user level.

However, the database access from the API violates the least privilege principle. Ideally, it should use differing levels of access for different requests, so that, for example, a fault in the code for handling a routine user request cannot accidentally invoke administrative-level changes. Barring that, the API at least should only have the permissions needed to do what it needs to do on a regular basis. Instead, the API layer uses an account with full, unrestricted access to the database.

Least Common Mechanism

Least common mechanism refers to separating the hardware and software used by different users, so that a security issue affecting one of them does not impair another. This concern is overkill for the majority of OpenMRS installations, but it could be valuable to have separate but closely-tied administrative and clinical OpenMRS installations so that the clinical operations can continue even if there is a fault affecting the administrative operations.

Psychological Acceptability: Grade A

Psychological acceptability refers to ensuring that the security mechanisms feel appropriate to users. OpenMRS has done an excellent job ensuring that security does not impair usability.

Isolation: Grade C

Isolation refers to providing barriers between users and resources so that only authorized resources can be manipulated. Most importantly, this includes ensuring that users cannot tamper with the security mechanisms. OpenMRS provides isolation only through the algorithms implemented in the API. It uses the same database connection for configuring security and for routine operations, which makes it likely that an algorithmic flaw could allow a regular user to reconfigure security. OpenMRS is also missing isolation to prevent clinicians from accessing records without a medical justification.

Encapsulation: Grade A

Encapsulation refers to using object-oriented design to separate the available functionality from the details of its implementation. OpenMRS consistently uses encapsulation.

Modularity: Grade A

Modularity refers to reusing existing, well-vetted security mechanisms instead of creating custom versions. This has been OpenMRS's standard practice.

Layering: Grade C

Layering refers to dividing an application into different layers, which facilitates complete mediation and can impair hackers by providing a new security barrier in each layer. All OpenMRS security is in the API layer. As a result, hackers can focus their attention on the API layer, and face no additional security checks in other layers.

Least Astonishment: Grade A

Least astonishment refers to having the application, especially with regard to security, behave in the way the user would expect. If the application does something unexpected, the user is more likely to react wrongly and make the problem even worse. The students consistently reported that OpenMRS behaved exactly as expected.

Recommendations

We recommend the following actions to address the most significant security risks for OpenMRS implementations. They are ordered with the most important recommendation first.

Force Change of Default Admin Password

The greatest security risk is that the default admin password will be exploited. Currently, this is controlled with emphatic instructions to change the password. Any of three straight-forward technical measures would eliminate this risk entirely.

Password Expiration

Add an expiration date to the table containing the user password. If the expiration date has passed, the user is not allowed to do anything else until the password is changed. Then, the default admin password can be given an expiration date in the past to enforce the recommendation that the user change it immediately.

There are a number of best practices to keep in mind while implementing this:

- The administrator should be able to set a policy for how often passwords expire. By default, passwords should expire once a year, which provides a good balance between security and usability. The administrator should be allowed to disable this feature, but of course only after changing the default admin password.
- Many systems have a second expiration date, at which time the user's account is disabled until an administrator reactivates it. This is generally about 30 days after the first expiration date. It would be good to implement this feature to provide more flexibility for local security policies.
- Many users try to keep their old password by changing it back immediately after it was just changed. The database should keep track of the previously used passwords, and prevent a user from reusing a password too soon. It is best to keep old passwords for a configurable period of time, defaulting to 13 months.
- Consider adding an emergency override for clinicians, in case the delay of changing the password would put a patient's life at risk. This feature is a double-edged sword, in that it could easily be abused. Consider allowing it a maximum of 3 times before changing the password, with no ability for the administrators to configure this, to minimize the interaction with local politics.

Password Quality

There are a number of open source projects that provide code for enforcing password quality. This should be incorporated so that the password is checked against the local policy every time a user logs in. The default policy should be strong enough that all default admin passwords are rejected. When a user logs in with a weak password, the user must change the password before being allowed to do anything else.

An emergency override option could again be valuable, with the same caveats.

Password Blacklist

Implement a list of unacceptable passwords, which should contain every password that OpenMRS has used as a default password and published lists of commonly used bad passwords. Every time a user logs in, compare the password to the list, and require an immediate password change if the password is on the list.

An emergency override option could again be valuable, with the same caveats.

Inactivity Timeouts

There should be a mechanism to lock access to the system if the user has not interacted for a period of time.

The API should keep track of when it was last accessed by a session, and require reauthentication after a configurable delay. It would be a good idea to let users pick a PIN so that they do not need to enter their whole password every time the session times out.

The reference application should track inactivity, and require entering the PIN to continue the session after a configurable timeout.

The reference application should use the HTTP response headers to disable local caching of content, so that the protected health information is not exposed.

The JavaScript in the reference application should track inactivity, and erase all protected health information from the screen after the configured timeout.

Excessive Failed Login Locking

Track the number of failed login attempts for a user, and lock the account after 5 failed attempts. The administrator should be able to configure the number of attempts before locking the account and the length of time that the account will be locked.

Auditing

A strong auditing system needs to be implemented to provide accountability and records of access to protected health information.

We recommend that the auditing system have the following features:

- The auditing should take place in the API layer.
- There should be a class representing auditable events. It should be capable of describing any event that is worthy of auditing.
- Most entry points in the API should create an auditable event and post it to the event system. Certainly the system must be able to audit security changes, authentication, any access to protected health information and any access that requires authorization, including attempts to access protected resources without authorization.
- The event system must be carefully engineered to be robust to handling huge number of incoming events efficiently, in the event an attacker tries to overwhelm the auditing system to hide important evidence of the attack.
- There should be multiple options for consuming auditable events. Typically, these would include writing to a file, writing to a database, transmitting to a remote host for additional processing and actively alerting administrators (page, email, text, computer display, audible alarm).
- There should be a mechanism for configuring which consumers should receive auditable events based on the type and content of the event. This mechanism should also be able to set characteristics of the event, including especially the data retention policy.
- There should be user interfaces for configuring the policies and viewing the audit logs.
- There should be a curator process that deletes auditable events from the archives as specified by the data retention policy.

Document Security Design Decisions and Implications

You should provide a section on security in the implementation guide, in which you discuss the security design of OpenMRS, and provide guidance on what kinds of additional security review may be necessary. Ideally, implementers would be able to review this section carefully to determine whether

they can safely implement OpenMRS in their environment or if they need to do their own security assessment.

Separate Privileges

Currently, OpenMRS only has one kind of administrative account, which has authorization to do anything. OpenMRS should split the administrative functions into categories, and create a separate administrative role for each category. The initial administrative account would then have all of these roles, and different sites could select to assign only a portion of responsibility to each administrator.

An auditor should be able to configure the auditing system and manage the audit logs.

A security administrator should be able to create and manage user accounts.

A data administrator should be able to manage the protected health information.

A system administrator should be able to manage modules.

Additionally, patients should be divided into various categories, so that users can be restricted to only operate on specific categories of user. Preferably, even administrators should be restricted to categories of users so that attackers who gain access to an administrative account can only get a portion of the data.

Conclusion

We have assessed the security of OpenMRS, with special attention to HIPAA regulations. We found that OpenMRS has some important security weaknesses, but that it has also laid a good foundation for remedying these weaknesses. Most importantly, the default admin password violates the principle of fail-safe defaults, and should be corrected by implementing a password expiration or password quality policy. Additionally, OpenMRS should implement session timeouts and account locking with repeated login failures. Although more difficult to implement, we provided some feature recommendations for a much-needed auditing system. Further, we recommended a section providing security guidance in the implementers' documentation. Finally, changes should be made so that administrative responsibilities can be divided across multiple administrators. We believe that these recommendations will make a meaningful difference in the security of OpenMRS implementations.

Reference Application Authentication/Authority May, 2015

Authors

Nicholas B., Alex C., Xiaocan D., Zachary D., Kunal K.

Executive Summary

OpenMRS has major security flaws that need to be addressed in order to successfully gain momentum in its goal to become a prominent force in free medical software. However, there are basic methods in place that do protect the system and its assets from various forms of attack.

Already in place are some methods of protecting patient data and preventing users from access to parts of the system they should not be allowed to view and edit. There are restrictions on what users can and cannot view based on their role in the system (for example, a nurse and doctor have different levels of access to data consistent with their roles). Adequate system response to failed logins and unauthorized access is present.

However, major security flaws are present as well. Proper employee training is completely necessary in order to ensure the safety of the entire OpenMRS system. Simply knowing a registered username and password will allow you to enter the system however, with no other forms of authentication needed. More fine tweaking is needed to completely round out the system in a security sense, but OpenMRS and its concept is a remarkable idea and important to the cause of open source products and the future in their respective markets.

Scope

Our team is working on assessing the web application of OpenMRS, specifically the authentication and authorization part of it. The federal Health Insurance Portability and Accountability Act or HIPAA was created with the goal to make it easier for people to keep health insurance, protect the confidentiality and security of healthcare information and help the healthcare industry control administrative costs. According to HIPPA, determining authentication applicability to current systems or applications, evaluating authentication options, and implementing authentication option all determine a web applications authentication.

OpenMRS uses custom mechanism for authorization and authentication for users in OpenMRS. We are trying to learn how secure OpenMRS login security is. The web applications nowadays normally use HTTP cookie-based authentication sessions, so users need to enter a username and password pair. This pair is then validated by the application against some internal user database. A session record is then created using the cookie set, which the browser will send with each subsequent request to the application. The application can then show data related to the authenticated user throughout their work with the application.

In order to complete our goal of assessment, we need to cover the parts of the application that deal with users and the systems assets that occur throughout the system. These are used daily by administrators and users of the OpenMRS system, so closing the gaps between the data and outsiders is especially important. Our team needs to cover any parts of the system that have to do with two different ideas. First of all, we should be identifying any place in the web application that could potentially be used to expose the user's data to hackers. Secondly, we should be trying to identify what could be done to tighten those gaps between the Web App and the API which uses user's data.

We want to check if lower level authorized can access any unauthorized information or can change their access role in anyway. It is immoral to do that because security and privacy can be challenging to achieve in Open source project. It gets easier for intruders when users can set their roles and access other people's confidential information.

Installation

When one of our team members first went to install the program, he had problems with the installation because he had Java 8 on his machine. He uninstalled Java 8, installed Java 6, Tomcat 6, MySQL and allowed write access to Tomcat 6 (which was giving him an error originally), and everything worked fine. He was then able to download the openmrs.war file in Firefox - chrome was downloading it in a zip file so it was giving him problems. Following the rest of the steps in the installation guide - https://wiki.openmrs.org/display/docs/Installing+OpenMRS, led him to figuring most everything out without major problems or searching. Proceeding through the steps on deploying OpenMRS, he was both able to get logged in to the installation wizard and set most things up without problems, as well as create and download the database along with updating the database. However, things were not working correctly even though he followed the guide step by step. He discovered that in order to get everything working, it was necessary to delete Tomcat 6, and MySQL fully. Then he had to download the standalone version of OpenMRS by itself which includes both Tomcat and MySQL, and run the jar file from inside there. After all of this, he was successfully able to run the program and move around in OpenMRS.

Assets

Username
Type of Asset: Data
Class: SEC
Value of Asset: Moderate
A username is a form of identification that allows a user to log into the system.
Threat Agents:
Hacker - potentially, stealing user data/system data
Malware - altering username/exposing username
Threats:
Hacker - Possible * Moderate
Malware - Possible * Moderate

Password
Type of Asset: Data
Class: SEC
Value of Asset: Major
A password is a value that matches with a username and allows access when matched with the correct username. It is highly valuable.
Threat Agents:
Hacker - potentially, stealing user data/system data
Malware - altering password/exposing password
Threats:
Hacker - Unlikely * Major
Malware - Unlikely * Major

Patient Registration Forms
Type of Asset: Data
Class: PII
Value of Asset: Major
A patient registration form is a form that requires personal data of a patient to identify them. Such forms consist of names, birthdates, address, phone number, etc.
Threat Agents:
Keylogger - could snag data being entered into the registration form
Shoulder Surfer - could see private information not meant for them
Threats:
Keylogger - Possible * Catastrophic
Shoulder Surfer - Possible * Minor

System Administration: User Management
Type of Asset: Data/Communications
Class: PII/SEC
Value of Asset: Major
A form that allows anybody to add a new user or change the users in the system, allowing anybody to have access if the basic username and password is discovered and not changed.
Threat Agents:
Hacker - Could change any sort of user in the system
Malware - Could change any sort of the user in the system based on the malware's code
Keylogger - Could snag information / updated information being passed through forms
Threats:
Hacker - Unlikely * Catastrophic
Malware - Unlikely * Moderate
Keylogger - Unlikely * Major

System Administration: Visits Management
Type of Asset: Data/Communications
Class: PHI
Value of Asset: Minor
Data that is of little value except for record keeping purposes. Only includes patient ID, name,

time of visit and time of departure.
Threat Agents:
Hacker - someone that would want to disable the visit record system
Threats:
Hacker - Unlikely * Insignificant

System Administration: Provider Management
Type of Asset: Data
Class: PII
Value of Asset: Major
Data that includes the name of the providers as well as their role in the medical facility. Very important data in a financial sense since this data is used by insurance companies to denote the practitioners in the medical facility.
Threat Agents:
Hacker - someone who can alter or steal the practitioner data
Malware - can alter roles and names of practitioners or even delete information
Threats:
Hacker - Possible * Moderate
Malware - Unlikely * Moderate

System Administration: Data Exchange Module
Type of Asset: Data
Class: PII
Value of Asset: Critical
A list of all the data in the system that can easily be accessed by anybody with permissions. If this got out it could be catastrophic.
Threat Agents:
Hacker - Anybody who wants important information of users and doctors.
Rival Companies - Anybody who wants to steal users for themselves
Threats:
Hacker - Likely * Catastrophic
Rival Companies - Unlikely * Catastrophic

System Administration: Find Patient Record
Type of Asset: Data
Class: PII/PHI
Value: Critical
A form that describes patients with their important information including their telephone number, address, diagnosis, vitals, allergies, name, birthdate and their age.
Threat Agents:
Hacker - Anybody who wants important information of users and doctors.
Rival Companies - Anybody who wants to steal users for themselves
Malware - Anything that can expose user data to anybody else
Threats:
Hacker - Likely * Catastrophic

Rival Companies - Unlikely * Catastrophic
Malware - Possible * Minor

MyProfile: Change Login Info
Type: Data
Class: PII
Value: Major/Critical
Data that includes username, password, name, and secret questions and answers. Identification data that can be changed quickly.
Threat Agents:
Hacker - could, after logging in, change password to limit access to only themselves
Rival Companies - could look to prevent access to high level administration in the facility
Keylogger - could snag important login information if a user changes that information
Threats:
Hacker - Likely * Major/Doomsday
Rival Companies - Unlikely * Major
Keylogger - Unlikely * Catastrophic

Risks

In order to find the defenses against most of these files, the team had to search through OpenMRS documentation that was included in the OpenMRS GitHub account, under openmrs-core/web/src/main/java/org/openmrs/web/controller/. We also interacted with the application to see how the code functions.

Extreme Risks

There are no risks at this time that we would categorize as being “extreme”.

High Risks

Hacker:

Controls --

If you have insufficient privileges when you attempt to log on, in handling the request, the program will produce an error telling you that you don’t have the right privileges to access the program.

If you log in as a certain user, you are restricted to only that user’s level of access.

Missing Controls --

In a hospital setting, it is rare for people to stay in front of the computer constantly. The system should not assume continuity of user for indefinite periods of time, but should have some mechanism for challenging the user to reprove identity at periodic intervals.

Encryption - keeping your information protected every step of the way.

Not necessarily limited to a certain user - anybody can access data, no matter who they are. (Setting viewing privileges.)

OpenMRS admin users violate the principle of *separation of privileges*. If an admin user is bribed or if a hacker gains access to an admin account, there is no limit at all on what the user can view or change. Some effort is needed to divide admin privileges between roles and to otherwise curtail the power of admin users. Generally, admin users should have several user accounts with different passwords, one for routine work and another for admin work.

Keylogger:

If the computer is infected with a keylogger, it is up to the administrator to clean out their systems.

Brute Force:

Controls --

The controls for this type of attack are similar in type to the controls for Hackers, in that the system protects against multiple failed login attempts.

The system will also produce errors that let the user know that an attempt to hack into the system was attempted.

If the user gains access to any sort of staff besides a system administrator, they are limited to a certain amount of data.

Missing Controls --

A system where if the user makes a certain amount of attempts which fail, then the system will block that user for a certain amount of time until they can try again.

Make sure that users don't use passwords that contain words or phrases.

Medium Risks

Password Insecurity:

People may write down their password, stick it on a sticky note, and keep it in their office. This is extremely dangerous because anyone can just sit down at the desk and sign in with those credentials. There is only one control in place for this risk and there is no good way to implement a control for this. Users should remember their passwords and create strong passwords to protect the system.

In order to prevent this from occurring, we should make sure that users either keep their passwords somewhere safe or somewhere that they can access easily but not where other people discover it, such as their phone or their tablets. If they have a powerful password as well as a good hiding place for them, their passwords won't be discovered as easily, making it harder for outsiders to discover that personal information.

Malware:

If the computer is infected with malware, it is up to the administrator to clean out their systems.

Rival Companies:

Rival companies could try to hack into a system or implement a malware attack on the system. No controls seem to exist to prevent against the malware attack but there are some controls to prevent hacking.

Controls --

If you have insufficient privileges when you attempt to log on, in handling the request, the program will throw up an error telling you that you don't have the right privileges to access the program.

If you log in as a certain user, you are restricted to only that user's level of access.

Missing Controls --

Making sure the person is who they say they are

Everything tied into a system that can identify the users of the system as accurate.

Provide documentation on setting up a firewall and configuring Tomcat to restrict access to internal IP addresses.

Low Risks

Shoulder Surfer

There are no controls in place for this risk, and there never will be. This threat does not need controls because the risk of it happening is so low that it does not need to be controlled. There is no way to prevent an attacker from viewing data assets over a user's shoulder other than proper employee training.

We looked through what areas of code we could find in the OpenMRS github account, and in doing so we seemed to be thorough in our search. We also tried to search through to find specific areas that we needed to focus on in the source code. There are many different relevant parts to the code, so we can't fully know what's happening everywhere. However, our examination

showed us that there are points in the code meant to prevent insufficient privileges, wrong passwords, and other common types of attack against users of a system and the patients. We tried to find code that focused only on certain types of attacks such as viruses and malware, but nothing was explicitly stated. We do not understand how code protects against malware but we used our best judgment.

Design Principles

Economy of Mechanism

The security measures in place in OpenMRS are simple and small. Therefore, it does a great job at economy of mechanism. Its security measure to prevent unprivileged users from accessing unauthorized forms works well. If you have the privilege, you can access it. If not, you cannot. If you log in with incorrect credentials, it will not allow you to enter the system. The code for these are simple and straightforward. Its simplicity could be kept while also increasing the overall security of the program, however.

Grade: A

Fail-safe Defaults

This principle means that the default configuration of the system should be secure, even if the in team installing it fails to follow some steps in the instructions. By default, mysql root password is empty. Leaving mysql administrative user password as empty is not advisable. OpenMRS's administrative default username/password is very simple too. The default username is: admin, and the default password is: Admin123. That is not safe.

Grade: E

Open Design

In this specific context, the designs of security mechanisms in web application authentication and authorization are very open and not secret at all. OpenMRS is open source, meaning that it is available to anyone who wants to view it. The fact it is open source means its design is also very open to the public. However, we are unsure how many experts review this design and give their opinions on it.

It was also not simple to navigate the code to find security mechanisms (for us, at least). OpenMRS does well in its open availability to anyone who would like to review it, but it could most likely improve its initiative to have it reviewed by experts or simplify the means to skim through the code.

Grade: B

Separation of Privilege

Separation of privilege means that no individual user should have access to all of the assets in a system. This is important to ensure accountability for administrative users and to limit the impact of a security breach. OpenMRS does a mediocre job in implementation of separation of privilege. For example, a system administrator does not needs access to all of the patient data all of the time. Normally, only clinicians need access to that data.

Grade: D

Isolation

OpenMRS does a good job with isolation, which in this context mainly means protecting access to the security mechanisms. In this instance, users in the web app should not view security mechanisms that they are not privileged too. In turn, users should also not be able to turn off these security mechanisms and not be restricted because of security mechanisms in place.

An example of OpenMRS successfully following this principle is the fact that users cannot view any of the security mechanisms in place. For example, they cannot see the code that secures their roles and access on OpenMRS and that is good. In additions, they also cannot turn off any security mechanisms in the GUI available to them in the web app.

Grade: B

Encapsulation

OpenMRS uses different classes in java to divide roles of users. Doctors and nurses have different roles than each other. Their role is defined by the things they can see or change. OpenMRS may lack in overall security but it gives different users their own different environment to work in. However, any user can access and change the information of a patient easily. A positive effect of encapsulation in this project is the fact that when things need to be changed in the code it is straight-forward to find the relevant part of the code. In open source projects like this one, this is very beneficial to contributors.

Grade : C

Least Astonishment

The web app in OpenMRS does well with least astonishment. Everything works the way you would think it would. Every action mostly consists of a button and a title, where the title is straightforward and does not require much thought to find out what it is. As far as in terms of authorization and authentication, the web app in OpenMRS does not contain much astonishment. The system uses a username and password system in which both fields are obvious as to what they are. Every time you are not authorized to do something, the system will notify you that you do not have access to do so in a straightforward message. It does, however, bring up some odd

text with the message, providing some code. This would astonish regular users who do not understand coding, but not in a way that impairs security.

Grade: A

Summary of Findings

Risks

Protection against brute force password attempt attacks was lacking. The system does not prevent users from trying to login more than a certain number of times, which would be a good way of prevention for that attack. There is no method to ensure the user is who they say they are, especially if the user has walked away and another person has taken over the application. Along similar lines, it is important to check that passwords are adequately complex.

Design Principles

Fail Safe Defaults needs addressing, since the initial database and application administrator passwords are always the same. The failure in separation of privilege, where admin users have unrestricted access to the system, results in a substantial risk if an admin account is abused.

Recommendations

Proper Employee Training

Threat(s) being addressed: Brute force attacks, Password insecurity, shoulder surfer, malware, keyloggers

For many of our significant issues with OpenMRS, proper employee training will help aid in the fight against the issues being exploited. For example, brute force attacks can be counteracted with proper employee training. If the username and password are not simple and easy to guess, brute force attacks become meaningless. Password insecurity can be solved by training employees to not only make strong passwords, but to not share them or write them down and share them in a public environment. Training via a security expert would be ideal, but many resources exist online to inform users of strong username and password practice.

A method to test password security is to have users take a quiz made by a security expert. This way, their exact password and username are not necessarily known by the person grading the quizzes but they will have an understanding of how safe it is. For example, a question in the quiz could be “Does your password have your name/birth date in it?”. Evaluations could be then handed back and proper training could follow.

In the case of the shoulder surfer threat, proper employee training is simple and straightforward. Simply tell your employees to be aware of their surroundings and to try to set up the view on their machines to be as private as possible. There is no viable way to check if users follow this training but it is nonetheless important.

Proper employee training also prevents malware, such as keyloggers, from accessing your system as easily as it would without proper employee training. Teaching users to not use the machines set up for OpenMRS for other activities, such as downloading music or visiting Facebook, will prevent malware from having as great as an opportunity in attacking the system via the web app. Keeping constant logs of web sites your users of your machines visit is vital in checking if this recommendation is successful.

Timeouts

Threat(s) being addressed: Hacker, Rival Companies

A form of preventing unauthorized users from accessing an OpenMRS machine's data would be a timeout. If a user stays logged in for a certain amount of short time and is inactive (say, 5 or 10 minutes), then the system should log them out. This is just an extra security measure that could prove useful to prevent a hacker from attacking directly on an OpenMRS machine by reducing the window for attack. In order to check if this recommendation is implemented correctly, you would leave a computer idle for the amount of time that is needed for a timeout and seeing if it responds correctly.

Lockouts

Threat(s) being addressed: Brute force attacks

OpenMRS should prevent a user from trying to log in continuously after a number of failures. A lockout would prevent a brute force attacker by slowing them down and logging their login failures into the system log. A way to check if this was successful or not would continually failing to login to a machine and then checking the log after the lockout occurs, seeing if the date and machine that was being locked out were logged correctly.

Fail Safe Defaults

Threat(s) being addressed: Hacker

It is critically important that the system not allow the installer to use the initial default administrator password. There are several ways that this could be addressed. We recommend implementing all of them, as they each provide a different contribution to security.

First, every password should have an expiration date. When a password is expired, must change the password before having any access to the system. The initial administrator password should be marked expired, so that the installer must change the password before trying out the system.

Second, there should be a blacklist of common bad passwords. For example, there are many published lists of common passwords like "123456" and "password." The default OpenMRS passwords should be added to this list, so that they cannot be used. When a user logs on, if the password is found in the blacklist, the user must change the password before doing anything else.

There should be a mechanism for ensuring adequate password complexity. Typical policies include requiring lowercase, uppercase, numbers and symbols. Adding a mechanism for administrators to configure the policy and having a default policy that prohibits “Admin123,” requiring a user whose password does not pass muster to change it before doing anything else.

Conclusion

In the program’s current state, it does not have much going for it in the way of security and protection of the patient data. However, there are basic ideas and aspects of the program that limit how an attacker can manage to get into the system. There are also methods in place to prevent outside forces from accessing different parts of the system if logged on a certain user, but if the system administration password is unveiled then the entire system could be under attack. In order to prevent this from occurring, there are multiple things that the developers could do in order to prevent any outside sources from entering the system, and to protect it.

Some of the responsibility should fall under the actions of the user, but there still are things that could be done by the developers in order to protect the system from further harm when an attack gets into the system. The system admin themselves should understand how the system works and install the proper software to protect OpenMRS machines further, acknowledging the main weaknesses of the system and covering the holes in security it possesses.

Reference Application Authentication/Authorization May, 2016

We are assessing the WebApp component of the OpenMRS application. This is what the front end users use on the field to input patient information and medical records into the system. This app communicates with the API layer; information from the database layer is delivered to the API in the form of objects, which are fetched or saved using services in the Webapp. The Webapp also offers additional functionality such as access to medical terminology and procedure names for data entry users who might not be well-versed with these aspects of medicine. We are trying to analyze the authentication and authorization features built in this app layer.

We are assessing the authentication features that come built in the webapp. We will be exploring whether the authentication features work as expected and whether we are able to identify any vulnerabilities or security flaws. We want to ensure we have complete mediation, and that all assets are well protected. We will also be exploring the authorization and access control features that come with the Webapp. We want to ensure good security practices are being followed. Properly built access controls will have the principle of least privilege implemented. Also, we want to ensure there is enough separation of privileges.

We started working with the administrator account to create new user accounts and begin creating one for a patient. We found the authentication in this Webapp layer to operate in the usual way. However, outside of authentication we noticed some unusual behavior. We discovered that sometimes while creating a new patient, even when an error is displayed (such as "location required"), it still creates a record of the patient. This violates the principle of least astonishment. If an error is displayed claiming data is missing, a new record should not be added to the database.

Also, we noticed that when we create new patients and try to access their page, the URL shows the primary key for the patient ID that is being stored in the database. This is a security vulnerability, as a threat agent can try to access other patients' records by trying different primary key IDs in the URL. Some sort of a random number generator should be implemented in the patient ID creator algorithm instead, to strengthen the security infrastructure.

While logged in as the administrator, we tried to create new users and assign them different roles to explore the access control features built in the Webapp. There was a list of roles that could be assigned to users, and each role had a list of privileges. The list of privileges is viewable under the Privilege Management section on the administration page. Only the Admin can assign or take away these privileges from a set of users if required. Admins can also create new users. Data Users have the privileges to add patients and access further information about their records, but not to modify records. We found that Data Managers have more privileges than Data Users, in that they are able to change patient records. Access Control seems to be properly built, as per our exploration. One feature that we found to be unusual was that you are allowed to take an existing patient and add them as a user for OpenMRS, assigning said patient one of the aforementioned roles and their corresponding privileges. This has the potential to be a security vulnerability, and could cause a data breach. If this functionality has no practical use, it should be removed or disabled.

Assets and associated threats

1.

- **Name of Asset:** Patient's personal information
- **Type of Asset:** Data
- **Class:** PII
- **Value of assets:** Major
- **Risk Factor: Extreme** (Likely * Catastrophic)
  Patient's personal information, which can include patient's name, SSN, date of birth, home address, etc., and can be used by threat agents to bring harm to the patient in many ways.
- **Threat Agent:** Identity thieves
- **Threat:** Anyone trying to carry out identity theft by using personal information, in say a fraudulent bank transaction.

2.

- **Name of Asset:** Prescribed Medication list
- **Type of Assets:** Data
- **Class:** PHI
- **Value of assets:** Moderate
- **Risk Factor: Extreme** (Likely * Catastrophic)
  Medication for certain conditions such as AIDS and cancer could be highly expensive and could offer monetary benefits to threat agents.
- **Threat Agent:** Anyone trying to access a patient's medical history records.
- **Threat:** Threat agents could try and find details such as physical address and prescription date to locate patients being prescribed high value medication, and might organize a physical robbery to steal the medications.

3.

- **Name of Asset:** Procedure List
- **Type of Assets:** Data
- **Class:** PHI
- **Value of assets:** Moderate
- **Risk Factor: Medium** (Unlikely * Moderate)
  Record that keeps tabs on a patient's medical procedure list, which describes their medical conditions and other problems.
- **Threat Agent:**
  Companies that sell medical devices
- **Threat:** Advertising firms could narrow down a vulnerable audience by the process of elimination and then harass them with spam mail and phone calls about their products. For example, if someone is diabetic and a company sells insulin pumps, they can use a procedure list to locate the records of a patient who received insulin shots and bulk mail them with their product catalogs, violating the patient's privacy.

4.

- **Name of Asset:** Doctor's information
- **Type of Assets:** Data
- **Class:** PII
- **Value of assets:** Major
- **Risk Factor: Extreme** (Unlikely * Doomsday)
  People can use a doctor to narrow down patients that have certain problems.
- **Threat Agent:** Terrorist Organizations
- **Threat:** In some parts of the world, there have been isolated events where medical professionals from organizations such as Doctors Beyond Borders have been kidnapped. As per the report, "Attacks on aid workers in the region are common, and U.N. staff came under attack this year."(Source http://www.cnn.com/2009/WORLD/africa/04/19/somalia.hostages/)

5.

- **Name of Asset:** WebApp credentials
- **Type of Assets:** Software
- **Class:** SEC
- **Value of assets:** Critical
- **Risk Factor: Extreme** (Likely * Major)
  The WebApp credentials can be used to access and interact with a patient's record or data.
- **Threat Agent:** Patient
- **Threat:** A patient who manages to obtain a Data Manager's credentials is able to change records. A patient can update, remove or add fraudulent records, add/remove conditions or procedures which can result in different billing charges.

6.

- **Name of Asset:** Field Laptop Or Field Hardware
- **Type of Assets:** Hardware
- **Class:** SEC
- **Value of assets:** Moderate
- **Risk Factor: Extreme** (Almost Certain * Major)
  Hardware assets such as laptops and cell phones can bring good money to thieves, and can also be used to find out information on patients or procedures that will be happening.
- **Threat Agent:**
  -An openMRS user
  -Robber
- **Threat:**
  -OpenMRS users who have physical access to steal hardware and try to sell them on the black market
  -Unscrupulous people trying to steal clinical hardware such as tablets or computers that are used on the field by clinicians and data assistants

7.

- **Name of Asset:** GPS in tablets
- **Type of Assets:** Security/Hardware

- **Class:** SEC
- **Value of assets:** Minor
- **Risk Factor: Medium** (Highly Unlikely + Moderate)
  Threat agents can get current physical locations of openMRS users if the openMRS Webapp would allow such GPS coordinate details to transmit.
- **Threat Agent:** Hackers trying to discover locations of doctors, patients or clinics.
- **Threat:** A threat agent can use this to leverage information out of someone. For example, if an attacker can get access to your GPS, he/she may be able to determine where you live because from 10pm to 8am you are in the same place not moving. The attacker can infer that is your home.

8.

- **Name of Asset:** User authentication code
- **Type of Assets:** Software
- **Class:** SEC
- **Value of assets:** Critical
- **Risk Factor:** High (Less Likely + Major)
  The user authentication code can be used to have access to restricted privileges.
- **Threat Agent:** Someone trying to access PHI without being detected
- **Threat:** Hackers can use this asset to find out information about the patient such as where they live, which medications they take, and even when their next scheduled doctor's appointments are. If they can bypass user authentication or login as an admin and have superuser privileges, they are able to access this information and possibly edit the accounting records to avoid getting caught.

9.

- **Name of Asset:** Medical Record History
- **Type of Assets:** Data
- **Class:** PHI
- **Value of assets:** Critical
- **Risk Factor:** Extreme (Possible * Doomsday)
  The medical record history keeps track of patients, so in an emergency doctors need to have access to this information.
- **Threat Agent:** Unscrupulous hackers
- **Threat:** Anyone can use DDoS attacks to ruin one of these organizations by forcing them to lose access to critical data for patients such as medications and illnesses, and can have potentially life threatening consequences.

10.

- **Name of Asset:** Billing and Payment info
- **Type of Assets:** Data
- **Class:** PII
- **Value of assets:** Critical

- **Risk Factor:** Extreme (Somewhat Likely + Major)
  Billing and Payment information can be crucial, since it can be used to get access to someone's banking account.
- **Threat Agent:** Identity thieves
- **Threat:** Thieves might be able to obtain banking information if they are able to access records containing a patient's payment information.

11.

- **Name of Asset:** Access to admin privileges
- **Type of Assets:** Software
- **Class:** SEC
- **Value of assets:** Major
- **Risk Factor:** Extreme (Least Likely + Doomsday)
  If a data assistant manages to obtain data manager privileges(or better), they are able to edit data for a patient. This feature is likely to be misused. They can take down the whole program with these privileges.
- **Threat Agent:** Internal employees
- **Threat:** If a threat agent manages to obtain root privilege, they can change access control and assign more permissions to another user than allowed.

12.

- **Name of Asset:** Access to Physician privileges
- **Type of Assets:** Software
- **Class:** PHI
- **Value of assets:** Critical
- **Risk Factor:** High (Rare * Catastrophic)
  If a threat agent manages to obtain physician access, they will be able to perform all the things a physician can.
- **Threat Agent:**
  -Unscrupulous hackers
  -Drug Addicts
- **Threat:** Thieves can prescribe themselves medications and sell it for profits or use them for personal purposes.

13.

- **Name of Asset:** Access to WebApp source code
- **Type of Assets:** Software
- **Class:** SEC
- **Value of assets:** Moderate
- **Risk Factor:** Extreme (Possible * Catastrophic)
  A Malicious version of webapp with a backdoor to all the data being entered
- **Threat Agent:** Hacker or rival company
- **Threat:** Since the framework for the webapp resides on the local servers, threat agents can try and replace the genuine webapp code with a malicious version of the webapp on local servers.

The damages will be widespread, since all users will be using the local version. This can result in a threat agent putting up whatever he/she wants on the URL instead of an actual webapp, while making it look like the actual webapp.

Risks

Extreme Risks

Identity Thieves Seeking Personally Identifiable Information

A patient's personally identifiable information is easily accessed as long as a user can log in to not only the Administrator account (which is easy enough given its simple username and password), but even one assigned the Clinician, Data Assistant/Manager, Provider, or System Developer role. This is understandable for a Clinician or Data Assistant/Manager, but seems unnecessary for the remaining roles.

Suggested Control: Plausible controls would include limiting the ability for a System Developer to be able to see the exact data that is considered Personally Identifiable Information for a patient, and to make Clinicians and Providers separate roles with varying privileges. Additionally, administrative credentials should be more secured, with a less easily guessable username and password, and accessible only to people OpenMRS trusts to be an exceptional admin.

Fortunately, they do not allow non-authenticated users, called "Anonymous", or Authenticated users to view this information, which is a good control to have in place.

Confidence: We are confident in this assessment because we have used administrative access to be able to directly view these roles and the privileges assigned to them. We have also created Clinician and Data Manager accounts to test this as well.

OpenMRS has a general understanding of how to keep personal information safe, but should exercise caution towards what roles may view patient information and how easily obtainable the Administrator account is.

Attackers Seeking Medical History Records

Control issues with this threat are similar to the above, in that an attacker is likely to guess the Administrator account's credentials due to their very common implementation. As such, it would also make sense in this case for Administrator access controls to be more secure, with more complicated credentials and trusted admins.

Anonymous and Authenticated users also cannot view this information, which is a basic control that should be in place.

Confidence: We are confident in this assessment due to our understanding of how to view roles and privileges.

Once OpenMRS more carefully considers who may view medical records and how accessible the Administrator account is, they will have much more control over this threat.

Terrorists Seeking Doctor Information

The easy access of the Administrator account is a recurring issue with the OpenMRS Webapp. Here, the account can be used to analyze Clinician accounts for malicious purposes, such as targeting specific patients or even attacking the doctors themselves.

High Risks

Threat no. 1

Name of Threat: Threat of users assigning unauthorized access to other users

Assessment method and controls found(2):

- **Control No. 1**

We first logged in as the administrator and created new user accounts and assigned them basic privileges. Then we went in the manage users section while logged in as the administrator.
This is the url that shows in the status bar of the browser for a certain user
http://unix.cs.plattsburgh.edu:8080/openmrs/admin/users/user.form?userId=5
When we try to change the ending of the url to a 'userId=6' it displays the page for the next user in the system based on primary key 6.
http://unix.cs.plattsburgh.edu:8080/openmrs/admin/users/user.form?userId=6
When we logout and log back in as one of the basic users and try to use one of those links, we get an error page saying:-

- **Control No. 2**

Another control they have in place for this threat is the "Alert System Administrator" button on the error page. This way if somebody managed to gain root access and changed other users privileges already, when that user finds out they can alert the administrator.

General Impression on how well OpenMRS controls this threat
We assessed that OpenMRS does an adequate job at controlling this threat.
Control that can be added to make this threat control better
In case an unauthorized user is trying to access other users access control page by replacing the primary key id in the browser url, a notification should be sent to the system administrator. This can avoid many other possible attacks that we have mentioned earlier

- **Confidence:**
  We conducting this part of the assessment with **high** confidence. We performed in depth analysis, by creating different authorization roles and then checking to make sure they do what they are supposed to do. Then we even tried to break the code by trying to obtain information based on the primary key and were unsuccessful.

Threat no. 2

Name of Threat: Threat of users trying to perform SQL injection attacks to access or alter/access to other users login data

Assessment method and controls found(2):

- **Control No. 1**

We you try to perform a SQL like query in the URL mentioned in the status bar, we observed some unexpected behavior. For example when we type in the following url when logged in as someone other than the system administrator
http://unix.cs.plattsburgh.edu:8080/openmrs/admin/users/user.form?userId=6replaceuserId=5
We get the following notification:
"Insufficient priviliges: edit provider"
Attached screenshot below:

This is different from when we are just trying to access the provider details. In that case the error page displays that "Insufficient privileges:get provider"

This clearly indicates that we were able to cause some sort of a query in the logic layer. Threat agents can use vulnerabilities such as this one

General Impression on how well OpenMRS controls this threat
We assessed that OpenMRS does an inadequate job at controlling this threat.
Control that can be added to make this threat control better In case an unauthorized user is trying to add or remove characters from the url page, an alarm should be raised in the accounting system alerting the administrator of the user trying to perform such actions. This should then be further reviewed access other users access control page by replacing the primary key id in the browser url, a notification should be sent to the system administrator. This can avoid many other possible attacks that we have mentioned earlier

- **Confidence:**
  We conducting this part of the assessment with **medium** confidence. We believe we performed an in depth analysis. Although we were not able to get an actual SQL injection attack to go through, we were able to expose unexpected behavior.

Medium Risks

Low Risks

Assessment based on Design Principles

Economy of Mechanism

- **Grade:** C

Economy of Mechanism means if the design, implementation, or security mechanisms are highly complex, then the likelihood of security vulnerabilities increases. Subtle problems in complex systems may be difficult to find, especially in copious amounts of code. Considering OpenMRS is an open source software, there are many developers working on different modules which can then be incorporated in the main interface. This can induce difficulties while integration. We feel as if there has been no attempt at simplicity, at least within the WebApp source code. Everything was extremely difficult to find, and when something was found, it was difficult to comprehend. However, the WebApp itself was very easy and simple to navigate. Even though some aspects contained a few bugs, everything was placed in a location where you would expect it.

- **Confidence Level:** We performed this part of the assessment with **medium** confidence.

Fail-safe Defaults

- **Grade:** B

The delivered product should be secured on arrival. Unless a subject is given explicit access to an object, it should be denied access to the object. Default access to an object should be none. On inspection, we found that on initial installation, OpenMRS came with great built-in fail safe defaults. It came with a variety of pre-set roles that are appropriate for everyday clinic use, all across the world. The access control is well implemented, assigning the correct set of permissions to respective roles. Something of concern, however, is that the initial administrator username is “admin”, and the password “Admin123”. These are credentials typically set by default with many devices and user accounts, and are therefore easy for any perpetrator to guess.

- **Confidence Level:** We performed this part of the assessment with **high** confidence.

Open Design

- **Grade:** A

Open Design is a concept that the security of a system and its algorithms should not be dependent on secrecy of its design or implementation. OpenMRS is an open source project, so anyone is free to look at the code and exploit vulnerabilities. Therefore they do have Open Design implementation.

- **Confidence Level:** We performed this part of the assessment with **high** confidence

Separation of Privilege

- **Grade:** E

The setting of restrictions such that it would require many different people to breach security. Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key. OpenMRS is hugely lacking when it comes to the design principle of separation of privilege. At least in the WebApp component. If a threat agent manages to obtain administrator's credentials they have ability to do almost anything in the system.

- **Suggested Control:** A 2nd administrator should be set up as a separate entity. In order to assign a user privileges both administrators would need to sign off. This will prevent attacks from happening even if one of their account information gets compromised.

- **Confidence Level:** We performed this part of the assessment with **high** confidence

Least Privilege

- **Grade:** A

Least Privilege revolves around the principle of giving a subject the least amount of privileges needed to do its job. When we look at OpenMRS, it has excellent access control built. Administrator even has the ability to create new privileges and assign them to new groups. So a Data Assistant can go about doing their daily duties and in case they make a mistake and need to edit a patient they will not have access to make that change. A Data Manager would have to sign in to make that change. Similarly in case a user needs to be deleted, a data manager would have to contact their system administrator to remove the patient from the system. Really well built controls in place to ensure mechanism of least privilege is implemented properly.

- **Confidence Level:** We performed this part of the assessment with **high** confidence

---

Recommendations

Alternative username or temporary password

The default administrator username and password being ‘admin’ and ‘Admin123’, respectively, violates fail-safe defaults. These credentials make the product, in this case OpenMRS, insecure upon initial installation. With a username and a password so translucent, anyone can guess these credentials, and thus gain access over and change any and all data.

Have either an alternative username to ‘admin’ or temporary password that consists of random letters and numbers e-mailed to the user upon initial installation. After the users gain access to their accounts, they should be prompted to create a more secure login username and password.

To test the resolution of this issue, users, old and new, should try to install the Webapp with a legitimate e-mail address. After the completed installation, the users should find in their e-mails a temporary username and/or password to the WebApp. These temporary credentials should allow users access to the App, where they should be prompted to change their credentials to something more secure than the original login information.

Conclusions

Considering its numerous threats and vulnerabilities, the OpenMRS system is somewhat peculiar. All the different risk factors that were mentioned can lead to major upsets for the OpenMRS user. By assessing the WebApp components of the OpenMRS system we were able to look through some of the vulnerabilities it has in terms of security. This application consists of many assets and confidential information that should be protected at all times. As we have seen, it is not very hard to access a lot of this information on the server. This would be like a gold mine to threat agents attempting to access critical information that can be used for profit or personal usages. Seeing how a user has the privileges to access so much, some may question the authorization and authentication of the Webapp security system. While exploring the Webapp, we noticed some unusual phenomena, like the violation of principle of least astonishment, weakness of the security infrastructure, least privilege principle, and more. As a group we looked at the access control of the WebApp and we wanted to ensure there was complete mediation in order to protect all assets from threats.

Although OpenMRS is not perfect, it is still an open source project so anyone can point out vulnerabilities to improve the system. There are a few bright sides about the OpenMRS system. It is a medical record system that is used all around the world. Even though there are a few bugs in the system, many of its principles are trustworthy. One of the design principles the Webapp executes best is the Principle of Least Privilege, where the subject is granted the least amount of privileges needed to do its job. OpenMRS has excellent access control built in. Only the Administrator has the ability to create new privileges and assign them to new groups. If there are any errors, a Data Manager would have to sign in to make that change. By implementing the Principle of Least Privilege, there is a lower risk of threats, and one would not have too much power over the system. On the other hand, it feels as if Complete Mediation is less effectively implemented. We were only asked to enter our credentials just once, and after leaving the page idle for a good amount of time the account was still logged in. This may lead to attacks where a user or a patient is able to gain physical access to this machine. There are many flaws in terms of this system, but in order to improve, more precautionary steps have to be taken in terms of security.

Reference Application Confidentiality May, 2015

Authors

Marie H., Gabrielle L., Robert L., Brigham T.

Executive Summary

The OpenMRS WebApp has some major security flaws concerning the confidentiality of its patient data. A hard to accomplish installation process based on outdated software, followed by a very insecure administrator credentials setup, are just the beginning of problems which need to be addressed. The application also runs on insecure java code, which could lead to the alteration, exploitation, or theft of patient medical information. Many crucial design principles such as Fail Safe Defaults, Least Astonishment, Economy of Mechanism, and Complete Mediation are not up to par in the WebApp and should be improved as soon as possible. Without attention to such areas the WebApp is vulnerable to attacks and does not meet the necessary security requirements under HIPAA.

Scope

Our Group is assessing the WebApp of OpenMRS in regards to confidentiality. The WebApp consists of the user interfaces and applications themselves, built upon the API and database. This is the part of OpenMRS where users enter patient information, which is eventually sent to the database.

This assessment will determine if the WebApp meets HIPAA's security requirements in this area. It is important that there are no points of access to patient medical information that do not follow HIPPA's standards. Only authorized individuals should be able to access this information through the WebApp. Security at this layer is important because it acts as a separation between users and the medical records database, and as a result unauthorized access from it to the database should be prohibited--whether it be from incorrect user input or from attackers.

Installation Process

We first attempted to perform a manual installation of OpenMRS using the instructions available on the Implementer Documentation found at https://wiki.openmrs.org/display/docs/Installing+OpenMRS. We ran into some issues during step 4, when we were asked to install MySQL; it required a root password which we did not have access to. Next we did a google search for "installing MySQL without root password" in an attempt to bypass this issue. However, after many failed attempts we discovered that there was a Standalone version of OpenMRS which included MySQL. The directions for this installation were found at this link: http://openmrs.org/download/. It was as simple as extracting the tar.gz to a directory and giving it a java command.

Unfortunately, the Standalone version required Java version 6, while the system had version 8 installed. We were able to launch an OpenMRS WebApp page in the browser, but it was only a list of errors regarding the Java version. Without root access, we were unable to perform a Java 6 installation. Fortunately, an administrator agreed to install the version of Java that we would need in order to perform our assessment.

Next, we had to search for the directory that had Java 6, and copy the path to it. We then followed the instructions on the OpenMRS site

(https://wiki.openmrs.org/display/docs/OpenMRS+Standalone) which told us to use a modified command for accessing a different version of java. We used the path to the Java 6 executable file, followed by the command: -jar openmrs-standalone.jar. Unfortunately we were still unable to launch OpenMRS; we kept getting the same error message as before about having the wrong Java version. For now we are using an OpenMRS Demo available here: http://openmrs.org/demo/ to proceed with the assessment, alongside the webapp source code.

Login Credentials

Type of Asset: Data

Class: SEC

Value of Asset: Major

Login Credentials are the users password and username. Users are required to authenticate in order to access OpenMRS..

Threat Agents:

- Unauthorized Users
- Hackers
- Other Employees
- Keyloggers

Threats:

- **Almost Certain*Moderate** An attacker gains access to the system by hacking a users account. If the username and password are set to the default "admin" and "Admin123" this would be very likely.
- **Almost Certain*Moderate** Someone writes down their password and sticks it under their keyboard, allowing another employee/outside attacker access to their account.
- **Possible*Moderate** Keylogger captures a users username and password.

Patient ID

Type of Asset: Data

Class: PII

Value of Asset: Major

The patient ID is a number used to identify patients in the system. With it you can search for a particular patient, merge separate patient information, search and manage patient appointments, etc.

Threat Agents:

- Hackers
- Users

Threats:

- **Possible*Major** A hacker is able to search the data base for a particular patient and access their medial information.
- **Unlikely*Major** An unauthorized user looks up private medical information for someone based on their id.

Patient Registration Information

Type of Asset: Data

Class: PII

Value of Asset: Moderate

When a patient is registered in the database, the user enters the patients full name, gender, DOB, address, and telephone number.

Threat Agents:

- Other Companies
- Hackers
- Employees

Threats:

- **Possible*Major** A hacker could search for, change, delete data of a patient with their id.
- **Unlikely*Major** Another company could take this information to target a specific user in an attack.
- **Unlikely*Moderate** The employee entering the data could change it or steal it.

Patient Medical Information

Type of Asset: Data

Class: PHI

Value of Asset: Critical

Once a patient is selected, the user can view past visits, appointments, allergies, vitals, diagnosis, etc. about that particular patient.

Threat Agents:

- Hackers
- Other Companies
- Insiders

Threats:

- **Possible*Catastrophic** Corruption, theft, or loss of information.
- **Unlikely*Major** User acquires patient data maliciously and sells it to a third party.
- **Possible*Major** An employee could tamper with medical data or gain personal information about someone.

Appointment Scheduling

Type of Asset: Data/Communications

Class: Other

Value of Asset: Moderate

With this function of the WebApp a user can schedule appointments for patients and providers, and view appointments for the day.

Threat Agents:

- Malfunctions between WebApp and other parts of the system
- Hackers
- Malfunctions in Hardware

Threats:

- **Rare*Major** A power outage would be an issue because all of the appointments are scheduled and stored electronically.
- **Possible*Moderate** Unauthorized user views appointment data.
- **Possible*Moderate** An attacker can change/view appointment information of a particular patient.

System Administration: Manage Roles

Type of Asset: Data

Class: SEC

Value of Asset: Major

From the system administration section, a user can manage the role privileges assigned to other users.

Threat Agents:

- Hacker
- Other Employees
- Poor Setup
- Users

Threats:

- **Possible*Major** An outside hacker breaks into the system and changes/deletes the roles of certain users.
- **Unlikely*Minor** An employee is given a role that allows them to much access to patient information and the rest of the system.

System Administration: Manage Permissions

Type of Asset: Data

Class: Other

Value of Asset: Major

This is where the permissions of a particular user can be deleted or set.

Threat Agents:

- Employees
- Hackers

Threats:

- **Possible*Major** An employee could change his permissions to have more to access data he shouldn't be able to have. This could lead to him taking advantage of having a higher power.
- **Possible*Major** A hacker could break in, change permissions and target certain people to give them more or less access to data.
- **Possible*Catastrophic** A hacker could break in, give themselves full permissions and alter/insert with any medical data they wish.

System Administration: Manage Users

Type of Asset: Data

Class: SEC

Value of Asset: Major

This is where users can be added to the system and assigned certain roles.

Threat Agents:

- Users
- Hackers

Threats:

- **Possible*Major** A user changes their own role to something higher.
- **Possible*Major** An outside attacker changes the roles of others in the system.
- **Possible*Major** A user is created with poor credentials, allowing attacks to be successful on their account.

Configure Metadata: Manage Forms

Type of Asset: Data, Communications

Class: Other

Value of Asset: Minor

These forms are for things such as admission and discharge, or more specific medical forms such as eye test reports or vitals. Some are built in and cannot be modified while others can be added to.

Threat Agents:

- Hackers
- Employees

Threats:

- **Unlikely*Insignificant** An employee could tamper with these forms.
- **Unlikely*Insignificant** A hacker could alter format of forms.

System Administration: Manage Modules

Type of Asset: Data

Class: Other

Value of Asset: Major

This is where a user can manage the modules on the system by adding, removing or starting a specific one.

Threat Agents:

- Other companies
- Hackers

Threats:

- **Unlikely*Major** Rival companies could try to tamper with the system functionality by changing the module configurations.
- **Possible*Major** A hacker could also come in and change the module set up in multiple different ways.

System Administration: Data Exchange Module

Type of Asset: Software/Data/Communications?

Class: Other

Value of Asset: Major

With this module information can be exported as well as imported into the system. Users must have the right permissions to access this data so security is critical.

Threat Agents:

- Hacker
- Employee
- Competitors
- Malware

Threats:

- **Possible*Catastrophic** A hacker could break in and either export patient medical data or import their own falsified information/malicious code.
- **Unlikely*Catastrophic** Rival companies could come in, take the data and use it for their own gain.

Change Login/User Information

Type of Asset: Data

Class: SEC/PII

Value of Asset: Major

This information includes things such as the persons name, username, password and answers to security questions.

Threat Agents:

- Keyloggers
- Employees
- Hackers

Threats:

- **Unlikely*Moderate** Once a keylogger or hacker has access to that information they can see all that persons information as well as change anything they want to make the information inaccurate.
- **Unlikely*Moderate** A current employee(or hacker) with certain permissions could see all this information and either change it for their own personal gain or to limit someones access on the website.

Risks

Extreme Risks

An attacker gains access to the system by hacking a users account, eg. the admin username and password are not changed from the defaults.

Controls Founds:

- Authentication and input validation are both present at WebApp layer (at least for the Authentication field).
- There is a function that dictates whether or not a user needs to change their password when they attempt to log in, however it isn't clear how it exactly works.

Suggested Controls:

- Immediately require users to change their default settings and information
- Frequently suggest changing username and passwords to users

Confidence of Assessment:

- Reasonably confident. There seems to be a somewhat extensive list of permissions controls which need to be set by an administrator, and there appear to be controls set for cleaning up user inputs/invalid characters.

A user writes their password down and sticks it under their keyboard.

Suggested Controls:
(Obviously for this section, these controls can't be 'installed' into OpenMRS, but documentation could come with suggestions for best security practices, or non-intrusive tips on the authentication page of the OpenMRS WebApp.)

- Training employees not to store important data in obvious locations of a system
- Frequent check of employee work stations,

A hacker is able to access database information by exploiting JavaScript at the WebApp level.

Controls Found:

- Deny information access at the WebApp level without authentication
- There is input cleaning implemented, this should help stop JavaScript abuse.

Suggested Controls:

- The 'dwr' folder in the source code seems to contain methods of converting between Java and JavaScript objects. Because this utilizes JavaScript, we assume it is going to be insecure, and it is advisable to try to implement it in another way.

A hacker could also potentially compromise/delete/steal data.

Controls Found:

- Only Superuser can alter permissions.

Suggested Controls:

- It isn't

Information becomes corrupted or lost either due to an attack or from hardware damage/malfunction.

Suggested Controls:

- Regularly back up data
- Automatically store and save User's data after every input on the users end. Set up a function to do so
- There should be documentation for good hardware handling practices.

A hacker or non-super-user is able to access/change user permissions.

Found Controls:

- Only allow non-super-user's accesses/changes to be approved/permitted by Super Users.

A third party is able to maliciously export medical forms.

Suggested Controls:

- Deny permissions of third parties being able to export medical forms
- Only allow Super Users to perform such a confidential procedure (This may cause problems with efficiency if doctors/clerical staff aren't able to access or send medical data to say... insurance companies or something.)

High Risks

A keylogger captures a username and password, as a result of a user installing third party software on the system computers.

Suggested Controls:

- Regularly monitor log files
- Relocate system log files away from their default locations

An unauthorized user looks up private medical information for someone based on their patient ID.

Found Controls:

- Super Users can set permissions and restrictions for other users.
- Datatable requests are logged, require permissions, and when they are returned, are cast as an integer to combat cross-site scripting attacks.

Suggested Controls:

- Require even authorized users a second level of authentication to enter or access private medical information.
- The WebApp controller contains many routes to the data layer, including direct routes to patient and provider data. This section of the WebApp source code needs to be secured extra tightly.

Another company could try to use leaked information maliciously.

Suggested Controls:

- Require all employees to sign a confidentiality policy contract to ensure the safety and protection of information at one's company, even if a specific employee no longer works for one company
- Train employees to properly close out and log out of databases containing vital information

An employee with access to the data could attempt to use it maliciously.

Suggested Controls:

- Require all employees to sign a confidentiality policy contract to ensure the safety and protection of information at one's company

Hardware failure could cause loss of electronically stored information.

Suggested Controls:

- Regularly back up data (Backups should be secure and protected, with highly restricted access.)
- Automatically store and save User's data after every input on the user's end. Control: Set up a function to do so
- Train employees to back up data after each session

A module could be compromised/changed/added to the system, to cause unintended behavior or create a backdoor into the system.

Suggested Controls:

- Require all employees to sign a confidentiality policy contract to ensure the safety and protection of information at one's company
- Train employees to properly close out and log out of databases containing vital information
- Flag unintended or erratic behavior within a system. Super User(s) should be notified immediately

Low Risks

An employee is given a role that allows them too much access to patient information.

Suggested Controls:

- Perform regular check up on employee roles to ensure proper permissions are set to the correct employee
- Require employees to report if their or other accounts are given these permissions.
- Log files are required to pick up on these malfunctions. If a patient's information is leaked and there is no record of another employee accessing these records, then someone clearly has unauthorized access.

Design Principles

Economy of Mechanism

This design principle is the practice of keeping things as simple as possible in regards to security mechanisms so that users are able to navigate throughout the website and accomplish their

specific tasks. The design of the WebApp scores high for this principle; navigating around the website was fairly simply since every tasks had a clear menu to click on, which brings the user to the assigned area to help fit the needs of the user. In contrast, during the installation process this principle was not present as we tried to install OpenMRS. We were faced with multiple problems as we tried to install the system--even though the directions appeared to be simple at first glance, in reality the process was quite complicated. However, if this system were being installed in a real hospital or health center, a computing professional would most likely be doing the install, making it actually very simple for the users.

Overall Grade: B

Fail-safe Defaults

Having fail-safe defaults is extremely important in any system, but they are even more so in one that has valuable, private information stored on it as OpenMRS does. Every OpenMRS installation comes with a default admin account with username "admin" and password "Admin123". This default is a huge violation of a good fail-safe default policy.

Overall Grade: C

Open Design

A system with open design means that the system is operated by common knowledge and just about everyone using it understands how it works. Security of the system should not rely on complicated mechanisms or the ignorance of a user, but rather on protected mechanisms such as passwords and keys. OpenMRS publically publishes its code for the system, so it certainly isn't relying on some complicated security procedure to deter attackers. It does have simple password based authentication, and the navigation of this website follows this design principle since it is quite easy to move around. However, there are not that many security controls in place to begin with so the openness may not be that as beneficial as it could be.

Overall Grade: A

Least Privilege

Least privilege requires each entity that uses the system to be given the smallest amount of privileges that are needed in order to get their work done. Each employee should have the permissions required to do their job and nothing more. This protects sensitive data, reduces the likelihood of malicious behavior, and enhances security. The OpenMRS WebApp has a good system for setting the privileges of each user and assigning them to a particular role. Administrators should carefully assign roles to new users and set privileges that are compatible with the users role, while at the same time not allowing more rights than are necessary.

Overall Grade: A

Psychological Acceptability

For this principal it is crucial that the security mechanisms do not hinder usability or the accessibility of resources. The interface should have high usability so that users do not try to circumnavigate the security controls and apply the controls correctly. The controls in place should match up with the expectations of the user. For example, in the WebApp when a user enters their username and password incorrectly they expect that an error message will be produced. It is also important that the message not display any unnecessary information that could benefit a potential attacker. The simple "Invalid username/password. Please try again." message produced by an invalid attempt is a good example of a correct security control that does not hinder the user's psychological acceptability, of the system. The WebApp scores high in this area; however, there are not actually many controls in place in this application.

Overall Grade: A

Least Astonishment

Least Astonishment refers to the fact that when a users uses the webApp it should behave in an obvious, predictable manner. If an outcome surprises the user then the application fails the least astonishment test. The installation of OpenMRS seems to be lacking in this area,, due to the amount of trouble we had when we were following the installation process. However, the overall layout of the WebApp interface is very clean and self explanatory and we encountered no surprises as we went through the demo.

Overall Grade: B

Summary of Findings

We were relatively happy with our inspection of the OpenMRS WebApp, with a few key exceptions, which will be outlined in further detail below. In short, the default admin username and password are inexcusably insecure, JavaScript raises some serious red flags in terms of security, and, with all the information accessible through the WebApp, we'd like to see security beefed up in a few places.

Recommendations

'DWR' in the WebApp Source Code

This part of the Source code utilizes AJAX DWR to translate between Java and more web-friendly JavaScript. It's important to validate all inputs while using JavaScript, which the WebApp seems to do reasonably well. Still, JavaScript is notoriously insecure, and because of this, parts of the WebApp may remain exploitable, even after good input validation. My recommendation would be to try to find an alternative way to support the functionality that DWR provides without running the risk of JavaScript vulnerabilities.

Extra Security at the Controller Level for the WebApp

The controller code in the WebApp source allows (necessarily) for a large amount of access to the database layer of the OpenMRS application. While user privileges attempt to mediate between the interactions between the WebApp and the database, I believe this part of the code would benefit from additional security controls, especially for installations with an internet connection. Perhaps something along the lines of encryption between the database and WebApp layers to deter anyone attempting to access information as it travels over the network would be a reasonable start.

Better Default Admin Username and Password

The default username and password for the admin account in an OpenMRS installation is currently not fail-safe. It is both easy to guess and not very complex, following a very typically username-password pattern. Though one would hope that it would be one of the first things an admin for an OpenMRS installation would change, there is no guaranteeing that they would, and this violates a fail-safe default policy. My recommendation would be to implement some sort of algorithm to generate a random password when OpenMRS is first installed.

Conclusion

OpenMRS has great potential to becoming a worldwide medical records service, but before it can evolve to that step it must first undergo change to establish strong security protection of the whole system. The WebApp of OpenMRS has a nice, clean look and is overall very user friendly. It includes at least an attempt a security managements, by using a role based system with certain permissions grated to each user/role. Theoretically, only authorized users should be able to change/access patient information in the WebApp.

Despite these positives, the WebApp does not satisfy the security requirements of HIPAA and patient information has he potential to be compromised by unauthorized users/outside threats. The installation process of OPenMRS was cumbersome and difficulty to follow in practice--something that might deter potential customers from purchasing the product. It also runs on an outdated version of Java, which clearly is cause for security concerns. Insecure coding practices in the JavaScript were found, as well as a lack of controls to protect data confidentiality. Another major concern was the easy in which an attacker could access the system with administrator privileges if the proper care was not taken to change the login credentials to something more secure at the time of setup.

Our audit has uncovered numerous security flaws in the WebApp, such as the ones mentioned above, that pose a threat to the confidentiality of the assets in the system. We strongly urge developers to consider the threats mentions in the previous sections as well as our recommendations, and use them to enhance the overall security of OpenMRS.

Reference Application Confidentiality May, 2016

Authors

Joe A., Alex C., Akshay S., Nhat V.

Executive Summary

This page details the security assessment of the OpenMRS web application. Specifically looking at the confidentiality aspect of the web application. The page discusses scope, assets, risks, design principles, and a series of recommendations based on the previous items. The web application is in need of some security updates. There are attempts to keep the confidentiality of the assets a sure thing, however it is our belief that these attempts are not enough.

The assets include patient information as well as user information. We recommend several changes including the introduction of more authentication levels to protect patient health data from those with no need to view it. The report below will go into more detail.

Scope

We will be assessing the WebApp of OpenMRS and its confidentiality. We will be taking a look at the various topics this includes, such as privacy of health information, user information, how forms are processed through the code and if it is secure, how the interface works etc.

My group will be going through the relevant source code of the WebApp to try to see how OpenMRS handles many of the possible confidentiality factors that can come into play. We will be trying to get the confidentiality aspect of OpenMRS up to par with HIPPA guidelines and to make its privacy stronger for its users and patients. After we have gone through the source code and getting an idea on how things work we will try the Reference Application and use it for ourselves to see how things can be improved and what works, what does not work.

After reviewing some of the source code stored on GitHub and using the Reference Application:

- **PseudoStaticContentController module:**This module has a possibility of being a confidentiality risk, it is quite nice for ease of use for the user since it stores relativedata straight to their internet browser, however it states at the top of the module that all jst1 files are cached until a serverrestart. From what I can tell this is the log files made by users. This log file could contain health information at risk. This could potentially risk it, which would go against HIPAA guidelines of private health information.

- **Liquiobase-demo-data:**From what I gather this is not a direct confidentiality feature, but it switches what was previously used as "Person_name" to "Patient Name." The change is interesting but also helps identify the patient better which helps the Identifiable guideline in HIPAA where the identity needs to be confirmed of the patient.

- **Lack of Source Code:**Now this may not be completely true when it comes to viewing the forms, submitting the data, but I have been unable to find a large amount of source code in the web/webapp directories of Git Hub that pertains to confidentiality in the WebApp itself. This is very interesting as it starts to make me think that there is very little confidentiality controls in place in the application. This could lead to major problems with HIPAA guidelines or in the general public of their patients who's health information is stored in OpenMRS.

- **Searching/managing for Patients/Users:**From using the interface application I have learned that you are able to search for any patient with no restrictions. I feel this could be improved upon, having certain doctors or nurses registered to a patient, and having them be the only ones with access to the health records.

Assets

Username

Type of Asset: Data

Class: PII

Value of Asset: Moderate

The username asset determines who is authenticated to login to OpenMRS if the password is correct.

Threat Agents:

- Inside threat from employees looking to gain access to confidential health information.
- hacker from an unknown source who is trying to gain access to a patients health information
- A hacker holding the system hostage once he is into an Administrator account

Threats:

- Possible*Moderate
- Likely*Major
- Likely*Major

Password

Type of Asset: Data

Class: PII

Value of Asset: Critical

A special phrase created by the user or administration that authenticates a username for logging into the OpenMRS system.

Threat Agents:

- A hacker trying to break into the system to steal information
- A hacker holding the system hostage once he is into an Administrator account
- Inside threat from employees looking to gain access to confidential health information.

Threats:

- Likely*Major
- Likely*Doomsday
- Likely*Major

Patient Information

Type of Asset: Data

Class: PHI

Value of Asset: Critical

Patient Information could range anywhere from names to diagnosis. Depending on local privacy laws the value of this information can range from insignificant to critical.

Threat Agents:

- Insurance companies trying to find out your condition.
- Pharmaceutical companies looking to target people with specific ailments with advertising.
- Hackers looking for valuable info to sell to third parties.

Threats:

- Likely*Moderate
- Likely*Moderate
- Likely*Catastrophic

Identification Number

Type of Asset: Data

Class: PII

Value of Asset: Major

The identification number relates to patients and their files located in the OpenMRS database system.

Threat Agents:

- If someone gets the ID Number of a patient they can look at their entire medical history.
- If an ID number is compromised the "hacker" can alter important medical information
- Hackers looking for valuable info to sell to third parties.

Threats:

- Likely*Major
- Likely*Doomsday
- Likely*Major

Provider Information

Type of Asset: Data

Class: PII

Value of Asset: Major

The Provider Information includes the name of the staff and their corresponding identifier number

Threat Agents:

- Hackers looking for valuable info to sell to third parties.
- Insurance companies looking for providers information.
- Other Hospitals looking for new employees.

Threats:

- Likely*Major
- Likely*Moderate
- Unlikely*Insignificant

Location

Type of Asset: Data

Class: SEC

Value of Asset: Moderate

The Location includes hospital IP address

Threat Agents:

- Hackers looking for ways to hack into the hospital system and attack other things

Threats:

- Likely*Moderate

Scheduler

Type of Asset: Communications/Software

Class: SEC

Value of Asset: Moderate

A scheduler is the piece of software put into place in OpenMRS that allows appointments and any other task that needs to be done or scheduled for a certain time can be logged into a calendar for appointments.

Threat Agents:

- Someone trying to find when a person is going to be getting their medication, so they possibly can alter it
- Someone finding out when/where a person will be at a specified time.

Threats:

- Unlikely*Critical
- Likely*Moderate

Risks

Place each threat into one of the following categories based on the table on p. 505 in Stallings.

Extreme Risks

Leaked Privacy Information

- To access certain information from the database, it seems all you need is an authorized account that can log into the web application.

- A missing part of control would seem to be privileges for accounts. When my group was taking a look at the web application we were able to create new patients and edit them without and sort of authentication. Whether this is only access for the root admin, or for everyone it still gives root admin unlimited power in the database. There needs to be some kind of control system to create limitations for every account that has access.

- They could make an encrypted password for the root admin and give parts of it to multiple different people which can make it nearly impossible to access the "unlimited" power of that account.

- I am fairly confident in this assessment because I used the Web Application we were given to test and spent a fair amount of time with it to test these features. I could have possibly overlooked certain things in the applications Administration page since there are many different things you may do as a super user.

- My overall impression is that openMRS has a system in place that can be very effective in doing what needs to be done in protecting private health information, however they need to find a way to make sure it is not easily accessible through anyone. They need to set up some sort of "Checks and balances" system where not just one person has complete power.

High Risks

Inside/outside hacker attack

- Throughout looking at the documentation and source code we have found very little that can actually help prevent and catch an inside attack on the openMRS system.
- User name and passwords help limit those without access to certain parts of the medical record system. This will reduce the probability of an attack being successful.

- Some controls that have not been found in any of the documentation is an audit/logging tool. This tool can be helpful to track down where and who the attack came from within the company, this can also identify an attack that happened in the past, figure out what happened, and allows repairs to occur because they can tell what damage has been done.
- Besides Authentication of user name and password there is very little security, no certain code given to employees to access the database. It seems that the only thing you need is an account.

- I am somewhat confident in my assessment as no source of auditing was found, and it was clearly stated in class that it is a tool the developers need to implement. I understood a majority of what was happening in the source for the Web App, even though there was not a whole lot to look at that regards confidentiality.

- My opinion is that this threat could very well happen, and if it does it could put lives at stake depending on what the hackers goal is while breaking into the system. Since there is no auditing, and the security in general seems lacking that anyone with some hacking experience could break into the system and cause havoc.

Medium Risks

Staff Information

- The only way I can tell that the staff information is confidential is the fact that certain user names are more than likely randomized. They are not specific to a persons name, although this

can entirely depend on the hospital that is using openMRS. The "admin" account is a prime example of how accounts come by default, but can more than likely be altered.

- Obvious controls are that you need an admin account to view and control user accounts, and admin accounts are only given to specific individuals that monitor and maintain the database. This will limit the probability of an attack on this information as they have to breach a more secured account.

- Some controls that may help this risk even more would be having administrative accounts that do not have full power. Instead of one account being able to view all employees, there could be certain specified accounts with the power to do so which can limit the account that are at risk.

- This is a similar risk compared to the health information remaining private which makes me fairly confident that the controls in place are what they should be and that there needs to be some sort of balance between admin accounts.

- Even if there is no limit on the admin accounts on whom can view the employees information, openMRS seems to have a good idea on allowing only admin being able to edit,create, view accounts with information like that...this will limit the potential danger of this information being leaked along with the amount of damage that can be done.

Low Risks

Location of patient/employee

- there is a tool called Scheduler which keeps track of patients coming in for a check up, when they visited the hospital, employee meetings etc...pretty much anything that goes on in the hospital. This is an invaluable tool that is included within openMRS because it keeps track on what is happening through the system, and facility.

- I feel like they could improve on the scheduler tool. They can make it more detailed and give exact details of everything happen. This will help clean up an attack, or repair what has happened because they know what was going on at that specific data and time.

- I am fairly confident in this overview of the location risk. This seems to be a major tool that is and was developed for this purpose alone. I spent a decent amount of time using and messing around with the tool in the web application to understand how it works.

- overall nothing is perfect but this tool is a great step in the right direction. I feel that this threat is handled very well, and should be of little concern to the developers and staff.

Design Principles

Economy of Mechanism

To keep the WebApp as simple as possible to prevent over complication leading to unnecessary errors

B: It has basic overall effort to maintain confidentiality with a basic layout that does what it needs to do without making it overly complex.

Fail-safe Defaults

To make the software secure from the moment it is installed for use.
C: The system when installed produces a very insecure user name and password for the administrator. Unless they change it immediately, it creates a huge security vulnerability for attacker to take advantage of.

Complete Mediation

The application of checking the authority of a request whenever a request is made to access an object.

B: The system has privilege management options available to the administrator.

Open Design

The security ideology of having your defences out in the open in confidence that attackers will not be able to breakthrough them anyway.
C: As it is an open source project, its security policies are open to the world to see. It has effective security but since it is not good enough for use in several countries, it could use some work.

Least Privilege

The strategy of giving employees access to only what they need access to in order to do their jobs.
A: The software allows the creation of different types of accounts for different users based on what their job is going to be. It gives them access to only what they have been designated.

Least Common Mechanism

The Strategy of preventing objects from using the same mechanisms to gain access to a resource
C: The software checks the same information to decide whether or not to grant access to a resource.

Psychological Acceptability

The security mechanisms should not interfere with the work of the OpenMRS users, but it should still meet the needs of those who authorize it.
C: Roles and privileges are assigned to different users that helps to block access to unauthorized users, but it doesn't interfere with the work of the authorized users.

Layering

Layering is one of the important item in Computer security Design of Principe where the designer have to secure every layer.

B. 'OpenMRS is designed to have tier architecture where the real strength is in its robust and flexible'.
According the our test-run and online research, OpenMRS's data model is built on API layer that allow developer to read/save to them.

Least Astonishment

The WebApp should apply this principle on their design and user interface. A WebApp without Least Astonishment will cause confusion and inefficiency to users.

A. The UI is very straight forward, leaves no confusion to our team to explore test-out.

Summary of Findings

- Throughout our security assessment of Web Application confidentiality some risks were handled very well while others are up for question. Some of the risks that were not controlled properly and are more at risk are the following: Inside/Outside threat attacks from hackers, and leaked privacy information. Both of these risks relate to each other by having type of hacker involved.
- These risks are both a victim of lack of user authentication. The hacker or user only needs to get into the account and they have clear access to multiple medical records and much more information. These need to be more secure and somehow have multiple ways to authenticate such as a user name/password along with an authentication code sent to a device that randomly generates a pass phrase.
- Some major design principles violated by just these two risks are integrity, least privilege, separation privilege, defense in depth, and confidentiality.
- Some other design principles are done really well. Such as the UI which is simple, straight forward to maximize the usability (Least Astonishment). We have also found that OpenMRS is built base on tier architecture where the users can scale up or down easily depend on their needs.
- Overall we found the app very reliable in gaining information about patients and users, but we must keep in mind that this is not always a good thing regarding confidentiality.

Recommendations

Change user permissions

- This recommendation relates to the fact that users can see pretty much anyone in the entire database by just either searching their name or their ID number that is related to them in the database. This hurts the confidentiality of patient information. One major way to fix this is lock each patients information to their own personal doctor/nurse staff. If more staff need to gain access, confirm with the patient if it is okay and simply give them permission to access the

patients health information. This would lead to a more privately held system, but with correct permissions you can gain access

Encrypt Data Export

- Data export allow the admin to download the patient data into a csv file. This file is downloaded without any encryption and it could be easily opened by anyone, if the admin loses it. This is totally against the Confidentiality part. It is very important to encrypt the file that has very sensitive information about the patients. The easier manual way to encrypt the file would be compressing it first and then adding a password on the compressed file. This would not allow the people to view the patient data if they don't know the password.

Conclusion

- The WebApp contains several assets that it needs to protect. These are patient and user data, including passwords, provider information, patient health data, identification numbers, and appointment schedules. The risk of these assets extends from the range of extreme risk to low risk. Even so they all require a level of security that can leave both patients and staff confident in the confidentiality of their information.

- The OpenMRS web application is in need of some security updates. There are attempts to keep the confidentiality of the assets a sure thing, however it is our belief that these attempts are not enough. There needs to be multiple levels of authentication rather than only having a user name and password system. Making changes like this will go a long way to making the system more secure overall.

Reference Application Accountability/Auditing May, 2015

Scope

The web application is the shortest route for hackers to infiltrate the companies most sensitive information. This is because the WebApp consists of interfaces that users enter input such as name and insurance information, which later is stored in the database. The task of securing a WebApp is both difficult and time consuming. We are going to find what application's security vulnerabilities that currently exist, walk through the auditing process, and identify high risk vulnerabilities. Essentially, we are going to assess the auditing capability of the OpenMRS WebApp. During our assessment we will be constantly checking if the OpenMRS is in compliance with HIPAA standards. We will also be suggesting areas that should be audited if they are not currently. It is absolutely vital that no unauthorized individuals have access to the information.

Installation Process

During our installation process, we ran into several roadblocks. We first started following the instructions provided at https://wiki.openmrs.org/display/docs/Installing+OpenMRS. We downloaded Java, TomCat, and SQL as directed to. After the downloads were successfully completed, we proceeded to the next step by downloading the actual OpenMRS which we located from http://openmrs.org/download/. We both have different operating systems so for convenience sake we chose the OpenMRS 2.2 standalone edition for Mac OS X.

When the downloads finish, we unzipped the folder and began to go through the contents. Inside contains a few files and other extended archive folders. After exploring some of these files and folders, we saw that in the README.txt file was instructions on how to open the OpenMRS. It instructed us next to open the OpenMRS-standalone.jar. We were on the home stretch, about to finally get the OpenMRS working when we opened a .jar file, which had two download options to available. We chose the "demonstration mode" presuming it could walk us through a step by step process of what to do next. Instead we got directed to a very ugly OpenMRS start-up error page. The error stated the version of Java we have downloaded was not yet compatible. We proceeded to uninstall the Java 8 we initially used and installed Java 7. After this we tried to open the OpenMRS from the .jar file again and we ran into obstacles even earlier than the last attempt. We also tried the OpenMRS 1.10.1 standalone version and still nothing have not been able to open the source.

In class we spoke with a group who had a similar problem with downloading the OpenMRS. We discovered a demo version existed which is basically identical to the actual OpenMRS. For the sake of time and prevention of further headaches, the remainder of our project we will be using the demo provided by OpenMRS.

Assets

Patients Login Information

Type of Asset: SEC

Value of Asset: Major

Description: The login information consists of the patients username and password which is required to enter OpenMRS.

Threat Agents:
Hackers
Unauthorized Users
Fellow Employees

Threats:
Possible*Major: Simplistic username and password could be hacked with little effort

Medical Records

Type of Asset: Data

Class: PHI

Value of Asset: Major

Description: Medical records cover a large scope of information including past appointments, allergies, prescriptions and insurance information.

Threat Agents:
Hackers
Employees
Family Member(perhaps a certain displeased mother-in-law)

Threats:
Possible*Major: information stolen and used by malicious hacker
Possible*Severe If an employee feels disrespected by a client they could take their information and alter it or keep it to try and get back at the client.
Unlikely*Severe If a client is released to early they will not get proper treatment.

Registration Process

Type of Asset: Data

Class: PII

Value of Asset: Major

Description: To sign up for OpenMRS, an account needs to be made. Similarly to making any type of account information such as social security numbers, phone numbers, current addresses and credit card information are required to create said account.

Threat Agents:
Hackers

Threats:
Likely*Severe If an unauthorized user is accessing this information they are most likely looking to get the information for personal advantage or to sell to another party.

Client List

Type of Asset: Data

Class: SEC

Value of Asset: Moderate

Description: This would allow access to a list of clients that one is not authorized to see.

Threat Agents:
Employees
Unauthorized persons attempting to access the computer

Threats:
Possible*Moderate: A person can steal a clients information for personal use or alter the list.

System Administration Privileges

Type of Asset: Data/Communications

Class: SEC/Other

Value of Asset: Major

Description: The system administrators control the permissions the users and their employees have.

Threat Agents:
Hackers
Employees

Threats:
Possible*Major: Hacker breaks in and takes control of the system, changing the permissions

Code

Type of Asset: Software

Class: Other

Value of Asset: Major

Description: This is the ability to execute source code on the web server as a web server user.

Threat agents:
Hackers
Web Site Administrator
Web Server User Process

Threats:
Likely*Severe If a hacker gets in, or the administrator is having problems with the company and wants to get them back, they can inject distinct codes to the facet script or gather private data through a weakness in the internet programs.

Application Failure

Type of asset: Software

Class: Other

Value of Asset: Major

Description: When a web application fails it is often a very public affair. This could come from a flash crowd, programming error, security hole that exposed data. These are all ways for the application to fail.

Threat Agents:
Software itself.

Threats:
likely*moderate The program most likely has coding holes in it or theres too much going on in the server to output properly.

SQL Attacks

Type of asset: Software

Class: Other

Value of Asset: Major

Description: SQL Injection is one of the many web attack mechanisms used by hackers to steal data from organizations. This takes advantage of improper coding of your web applications that allows hacker to inject SQL commands into say a login form to allow them to gain access to the data held within your database.

Threat Agent:
Hackers

Threats:
Likely*Severe SQL injection is a great way for hackers to pass SQL Commands through a Web App backened by the database. web applications may result in SQL Injection attacks that allow hackers to view information from the database and/or even wipe it out.

Design Principles

Economy of Mechanism

This principle deals with the design of security measures in both hardware and software. The goal is for users to smoothly navigate throughout the webpage with minimal issues. Simplicity is the primary goal for user satisfaction. The WebApp itself holds true to this principle. The webpage is straightforward and will please all users. The security, while it could be improved, is good. However we had a great deal of difficulty with the installation process. The OpenMRS does provide detailed instructions about the installation process, but they fail in this principle. With the multiple different programs necessary to install OpenMRS, it makes everything extremely complex. If you downloaded an outdated version or a updated version that is not yet supported, everything fails. But in a professional setting, one would assume an expert would be hired to install OpenMRS to avoid this issue.

Overall Grade: B

Open Design

The Open Design principle is concerned with how the design of a security mechanism should be open rather than secret. In the case of OpenMRS, the entire program is open source. Any person whether user, employee, administrator or a random viewer has the ability to view the code because its public. The Java script the WebApp is written in is included. We are not terribly strong at deciphering code so it couldn't hurt to OpenMRS to have an expert edit and simply the code to make it neater. Also for security purposes having all the code readily available could be detrimental if hackers see it so more security controls could be beneficial.

Overall Grade: A

Separation of Privilege

This principle deals with multiple privilege attributes that are required to achieve access to a restricted source. This is to prevent all users from abusing the system malicious intent. Privileges vary depending upon your position within the system. For the WebApp, it seems that each position simply requires a username and password and you have full access to OpenMRS. As long as no hackers compromise a high level account, the distribution of permissions is handled very effectively. And as always there could be more security controls for user piece of mind.

Overall Grade: B

Least Privilege

The Least Privilege principle makes sure every process and every user of a system should operate using the least set of privileges necessary to perform the task. Users have just enough control to complete their job. This is to prevent sensitive information to get into dangerous hands. This principle and the Separation of Privilege work hand in hand. The system currently in place for OpenMRS is very successful assigning the appropriate permissions to users.

Overall Grade: B

Isolation

The Isolation principle can be broken down into three parts: First, public access systems should be isolated from critical resources to prevent disclosure or tampering. Secondly, the process and files of individual users should e isolated from one another and kept where explicitly desired. Lastly, Security mechanisms should be isolated in the sense of preventing access to those mechanisms. An example in OpenMRS is how each medical record is stored separately under each patients name so mass medical information can not be hacked at a single time. Since the WebApp is the shortest path to the database, it is crucial that the isolation principle is followed. However, there is very little done to protect isolation. All a hacker must do is get a user name and he has access to patient records.

Overall Grade: D-

Encapsulation

Encapsulation is a specific form of isolation based on object-orientated functionality. Protection is provided by encapsulating a collection of procedures and data objects in a domain of its own so that the internal structure of a data object is accessible only to the procedures of the protected subsystem, also known as as designated domain entry points. Basically it combines a set of instructions or permissions into one large group. OpenMRS does an adequate job in encapsulating privileges by dividing users into groups such as users, employees, doctors and administrators.

Overall Grade: B

Layering

Layering uses multiple, overlapping protection approaches addressing the people, technology and operational aspects of information systems. By using multiple, the breach of one mechanism will not leave the system completely unprotected. It will not prevent an attack, but it will create obstacles to slow hackers down and give more time to stop he attack. In the WebApp, OpenMRS has broken down the source code into multiple layers which in our opinion is the best security measure they made in the WebApp.

Overall Grade: A

Least Astonishing

Least Astonishing deals with a program or user interface and how it should always respond in a way that is least likely to astonish the user. The user must have a good understanding of the program so the interface must be simple and user friendly. The WebApp in OpenMRS holds true to the least astonishing principle. Everything is straightforward and easy to follow, with little unexpected errors. Occasionally a link might be lacking information, but OpenMRS provides plenty of resources through links that answer most questions.

Overall Grade: A-

Summary of Findings

Applications provide the interface between our most sensitive assets, our data, and our users. It is of utmost importance that applications exercise the proper controls when it comes to protecting the confidentiality, integrity, and availability of data. Auditing an application can seem like a difficult task, but breaking it up piece by piece makes it achievable. From an auditing approach, to secure an application you need to go through proper data validation, protect interfaces first, and work with developers to achieve a more secure code. There are plenty of reasons for vulnerabilities, but lack of security awareness and application security training is one of the main causes.

Conclusion

Just like anything else, OpenMRS has pros and cons to it when assessing a Web Application Audit. It is essential for medical records to be to kept in confidentiality with the patient and staff that provides care to the patient. All the different types of attacks discussed in the audit shows that before anything there needs to be improvement in the security systems that can keep these records more confidential. It is also difficult to get the OpenMRS to open, there are some detailed steps and procedures including downloading java 6, anything later than that is not compatible and will not run. This could hurt them because most people like to stay updated with the most recent software technology. Another problem was the security login credentials. Attackers can easily access administration roles which can be very harmful especially if they change a patients information like their medical dosage.

Reference Application Accountability/Auditing May, 2016

Executive Summary

The WebApp should keep auditing everything that the users do because this will provide the opportunity for storing important sessions and actions that a user takes part in. Things that the user does may or may not be important for future reference but it can only help in the long run. Since this involves medical records and potentially sensitive information, auditing this App would be of great use if something were to occur and one needs to look back to see which user did what. It works like a security camera in a store. It helps to pinpoint the exact details of an event.

The auditing in the WebApp should enforce most of the design principles such as least privilege and separation of privilege. By using least privilege, this removes the chance that a user may do something harmful by accident. Separation of privilege spreads out the permissions giving less opportunity by a collaborative attack. These rules should not only be applied to users but to the actual audit system to ensure that the system is to corrupted or incorrect.

If these principles are applied, it may reduce the chances that vulnerabilities are exploited through threat agents. Threat agents such as hackers pose a threat in many circumstances and should be considered seriously when implementing the security of the audit system. Small preventative measures could help stop the threat agents from causing major damage to the overall system.

Scope:

We are assessing the Web Application and more specifically the code and it's initial settings when OpenMRS is first installed. Due to the fact that many programmers came to together to develop this software they've done a great job documenting all of their work thus far and in the source code all the files are in the correct locations and are easily accessible. There are however instances where the code works efficiently but can be hard to understand because it's all written in ajax. Since some of these programmers just wanted to get the job done it is coded densely leading to third party observers having a hard time understanding the code at first glance. One thing to note is that the Java is well organized and more thought seems to have been put into those files. This can actually help with security in the source code because it would take a large amount of effort for someone to access these source files when attacking your Web Application. Standard methods of guiding user input are well thought out as modes of auto complete and suggestion are there for users to add simplicity to the application.
In many files there are little to no comments which makes the code look nonsensical. An example of this is the file openmrs-core/webapp/src/main/webapp/WEB-INF/view/scripts/jquery/jqModal/jqModal.js which has no comments and it's hard to even understand what it does. We found out that this file handles pop ups to alert users of conditions in the application after further inspection.
https://wiki.openmrs.org/pages/viewpage.action?pageId=7111407 Most tables in OpenMRS include auditing attributes.
These include:

- changed_by
- date_changed
- voided _by
- date_voided
- void_reason

As found in the wiki,the audit information is automatically saved on objects whenever data is altered because of RequiredDataAdvice/SaveHandlers and/or the AuditableInterceptor hibernate class.
On this Wiki page it is discussed that there were plans to move to a centralized auditing system but I don't think that the designers followed through. The designers ended up just sticking with their original plan described above.

The Web App has a preinstalled featured called a yeoman generator which installs a framework for building your Web App logically assigning the environment based on specifications provided during the installation. Ultimately the final setup depends on the user but the scaffold set up by the yeoman has many failsafe defaults that are useful when starting up a web app with OpenMRS. There are security attributes already set up when yeoman is run giving the user a good frame work to start with and boosting the level of security in the web application. Although this is a very nifty feature it still has its fair share of problems. For one it still requires users to add key features to the application and with this comes inevitable user errors.

Any XML code in the source code is well thought out with many contributors on each file. There's even instructions in the source code on what to do if you want to change anything and how to get it to run properly after some tinkering. It should be noted that comments are handled properly.

Assets:

Username

Type: Data

Class: SEC

Value of Asset: Major

Threat Agents: Hacker, Logger Malware

Description: Username and passwords allows users to log in to the site.

Threats:

- A Hacker could get into a users account and change or view information. They could get the password to an account and view sensitive information.

- Malware could collect user passwords and and usernames and use to casue harm to the system.A Malware Web Logger could collect the username and password or a user.

Threat Probability: Likely, Moderate

Password

Type: Data

Class: SEC

Value of Asset: Major

Threat Agents: Hacker, Logger Malware

Description: Passwords allows users to log in to the site.
Threats:

- A Hacker could get into a users account and change or view information. They could get the password to an account and view sensitive information.

- Malware could collect user passwords and use to cause harm to the system. A Malware Web Logger could collect the password of a user.

Threat Probability: Likely, Moderate

Patient Medical Information

Type: Data

Class: PHI

Value of Asset: Moderate

Description: The patient medical information contains medical treatments, treatments and medication.

Threat Agent: Hacker, Inside Attack

Threats:

- Hacker uses social engineering to obtain information.

- Insider reveals documents to various sources.

- Moderate, Likely

Website

Type of Asset: Software

Class: Other

Value of Asset: Major

Description: The actual display of the OpenMRS website is and actual asset by itself. The website informs the public and possible users the purpose of the OpenMRS project.

Threat Agent:

- A hacker using cross-site scripting could potentially inject a script into the website and change the mission statement
- A hacker could potentially rewrite the instructions on the website for making a new login account and make you send the data to him/her.

Threats:

- Likely Moderate
- Likely Major

Identification Number

Type of Asset: Data

Class: PII

Value of Asset: Major

The identification number is a reference number linked to a patients information located in the OpenMRS database.

Threat Agent:

- A hacker trying to find who's Identification number matches withe a persons username.
- A hacker switching someone's identification number so they receive someone else's medication
- A hacker trying to sell someone's medical information

Threat:

- Likely Major
- Likely Moderate
- Likely Major

Server Hardware

Type of Asset: Hardware

Class: Hardware, SEC

Value of Asset: Major

The physical Hardware of the system containing the WebApp and its data. May be on site or remotely accessed.

Threat Agent:

- Theft
- Saboteur
- Fire/Flood/Weather

Threats:

- Thief steals hardware and sells for money

- Moderate, likely

Web App Network

Type of Asset: Data, HW

Class: SEC

Value: Moderate

The path taken to navigate to the website through all applicable infrastructure. This includes both internal and external networks.

Threat Agent: Hacker, Power, Malware

Threats:

- DoS/DDoS
- Packet Sniffing/Spoofing
- Service Outage

Design Principles

Economy of Mechanism

Economy of mechanism is a design principle which states, it is more beneficial and more secure for a design if it is not complex. It is more secure because of the simple fact that you can check if the code actually works and does what it suppose to. In the case of the OpenMRS software we were studying, the website was somewhat simple to navigate through. However, since OpenMRS is made up of a collective of inputs from different programmers all over the world, the website sometimes has a difficult time keeping to similar layouts for the user. This makes the user have to learn how to use different parts of the website differently because it was set up different.

Grade: C

Confidence: We would rate this a medium confidence.

Fail-safe Defaults

Fail-safe Defaults is the secure initial installation of the program. In the case of OpenMRS, the initial installation was secure and went on without a glitch. The only thing which we found could use some improvement is the username and password for the admin. It's really simple for a user to guess the username and password at the initial installation.

Grade: B

Confidence Level: We felt the initial installation was secure and have a high confidence.

Separation of Privilege

Each individual user of the Web App has their own account by which they access it. The auditing program logs and maintains a record for each activity taken under the individual accounts in addition to the records of what access rights the user has to the system. Individual accounts have access to differing scope of activities based on the level and requirement of each user; staff gain privilege based on function within the organization. Access to the database through the Web App is limited based on the function level of a user. Access rules are selected at the creation of the account and may be modified by the administrator later; default rules are done on a request basis

C. Conceptual privilege assignment through the web app "Request Account Manager" is handled well and there is a clear distinction of each privilege and access class - though what scope each has is not defined. The users are not automatically granted access until approved which limits the chance of fake account creations on a large scale; impersonation of an individual staff member is still possible.

Least Privilege

The users are given only the permissions that are necessary to use the app. They are not given extra permissions that they do not need. Records are kept showing what the user is able to do and only what they can do. Each account can only do what they are allowed to do and nothing more. If an account was authorized to do only one thing, the program will record what was done using this one privilege. Administrators will decide the limitations of the privilege that is assigned to a certain group. Whatever group an account is assigned to has limited permissions that are unique to that specific privilege.

Grade B. The limitations that certain groups have help prevent users from unintentionally changing something that wasn't meant to be changed. Using the WebApp could result in the corruption of certain services, but if the users are only granted permissions to carry out basic things and aren't granted anything else, this could save the system from damage.

Least Astonishment

The WebApp should respond to the user in a way that is least likely to surprise them. The software should respond appropriately to the input of the user. It should respond in a way that fits and is easy to make sense of by the user. If the outcome is unexpected, the user could become confused and frustrated with the app and turn to a different alternative.

Grade A. If the user were to click on the login button, the app should bring them to a page where they can log in with their credentials. If the user were to click on the login button and the site were to redirect them to a FAQ page with the login at the bottom of the page this would not be an appropriate reaction. The user would expect a page where they can clearly see where they are supposed to enter their credentials. Most login pages contain not much more than a place to enter the username and password of the user.

Conclusion

In conclusion we think that the Web App has a great record with security even though it's an Open Source Application. With all of the Source Code available online, there's only so much that can be protected because attackers could potentially look at everything. It's also important for users to keep track of every change made so that whenever an audit is performed on the system, it's very easy to view when and where a change was made.

API – Authentication May, 2015

Authors

Leonardo Alves Miguel (OpenMRS ID lmigu001), Lucas Narciso (OpenMRS ID lucasnar)

Executive Summary

OpenMRS offers a good authentication system based on the role-based authentication paradigm. However, there is visible place for improvement in the authorization system concerning verification of privileges in several places of the system that it is still lacking. It is also difficult to detect authentication attacks because there is almost no place for auditing of authentication attempts.

In order to state that, the API code was analyzed and the flaws discovered are explained in this document. The changes required to solve these flaws are not expensive, and it is not a very hard task to implement it. Some changes in the code are needed to follow better the Design Principles and the strategies we propose to deal with the risks. OpenMRS can be a stronger system in terms of security, allowing far more people to be helped, which is the goal of this open source software.

Scope

Our team will do a review of the authentication aspects of the API, as well as the access control policies, concerning data related to treatments, medical conditions, charges or payments (PHI). The API allows developers to interact with the complex OpenMRS Data Model (available at) with common Java objects. Our work will be based on source code review, without realizing the complete installation of the API.

The aspect of OpenMRS that will be the focus of our work is mainly the management of the privilege system related to the Services. The goal is to verify if each Service has appropriate implemented systems for authentication and authorization, assuring least privilege and separation of duty policies. We know that it takes place at the Context class, and at the Service Interface declarations. So, we will lead a review of the specifics of how this is working.

We will also check the session system to assure that the authentication and authorization remain secure during all the lifetime of a session. Once every request is wrapped in a session, it is important to assure that deception or intrusion is not taking place.

Installation

We have just copied the source code following the instructions at https://wiki.openmrs.org/display/docs/Using+Git, cloning the repository. All ran smoothly, as expected. We also tried to follow the steps at https://wiki.openmrs.org/display/docs/Step+by+Step+Installation+for+Developers to install the

API, but we got some errors when trying the installation (concerning something about XML files missing). The professor asserted that we do not have time to get it to work, so we did not get around the installations' issues, and this is the reason why we are going to do just the review of the source code.

Assets

Context.java

Type of Asset: Software

Class: SEC

Value of Asset: Major

Represents an OpenMRS Context, which may be used to authenticate to the database and obtain services in order to interact with the system.

Threat Agents:

- Insiders
- Hackers

Threats:

1. **POSSIBLE*MAJOR** An insider or hacker using it to gain superuser rights to the database or services.
2. **UNLIKELY*MAJOR** An insider or hacker using it can become any other user in the system.
3. **UNLIKELY*MAJOR** A hacker using it to forge a session with inappropriate rights.
4. **POSSIBLE*MINOR** A hacker using it to shutdown the system and provoke an accessibility threat.

Daemon.java

Type of Asset: Software

Class: SEC

Value of Asset: Major

This class allows certain tasks to run with elevated privileges. Primary use is scheduling and module startup when there is no user to authenticate as.

Threat Agents:

- Insiders
- Hackers

Threats:

1. **UNLIKELY*MAJOR** An insider or hacker using it to run tasks and sessions with elevated privileges.

UserContext.java

Type of Asset: Software

Class: SEC and PII

Value of Asset: Critical

Represents an OpenMRS User Context which stores the current user information.

Threat Agents:

- Insiders
- Hackers

Threats:

1. **POSSIBLE*MAJOR** An insider or hacker can use it to get superuser access.
2. **POSSIBLE*MODERATE** An insider or hacker can use it to provide false information about the user.
3. **POSSIBLE*MAJOR** A hacker with access to an instance can get sensitive information about the user.
4. **POSSIBLE*MAJOR** A hacker with access to an instance can get inappropriate privileges and then have access to improper roles.

ActiveListServiceImpl.java

Type of Asset: Software

Class: SEC

Value of Asset: Moderate

Default implementation of the active list service.

Threat Agents:

- Insiders
- Hackers

Threats:

1. **POSSIBLE*MODERATE** An insider or hacker can use it to add an improper service to a session.

Service implementations

Type of Asset: Software

Class: SEC, PII, and PHI.

Value of Asset: Moderate

Default implementation of the services.

Threat Agents:

- Insiders
- Hackers

Threats:

1. **POSSIBLE*MODERATE** An insider or hacker can use it to modify a service.
2. **POSSIBLE*MODERATE** An insider or hacker can use it to get sensitive information from the system.

PersonService.java

Type of Asset: Software

Class: PII

Value of Asset: Critical

This asset is a contains methods concerning Persons in the system. It can delete, change the attributes, etc., of a Person.

Threat Agents:

- Accidental internal agent.
- Intentional internal agent.
- Intentional external agent.
- Virus.

Threats:

1. **LIKELY*MODERATE** Deletion or change of one or more person's PII. If it is unintentional, then it is likely that not much data will be harmed.

2. **POSSIBLE*MAJOR** Information change or deletion by a internal or external agent. Harm could be done to many Persons, and this could be very harmful.
3. **POSSIBLE*MODERATE** Information being stolen by a virus.

PatientService.java

Type of Asset: Software

Class: PII and PHI

Value of Asset: Critical

A Patient is an extension of a Person. This class can create and change a new Patient. It also works with the patient identifiers (get, set, remove, etc.).

Threat Agents:

- Accidental internal agent.
- Intentional internal agent.
- Intentional external agent.
- Virus.

Threats:

1. **LIKELY*MODERATE** Deletion or change of one or more Patient's PII or PHI. If it is unintentional, then it is likely that not much data will be harmed, but in this case some sensitive PHI is present, so the harm can be worse.
2. **LIKELY*MAJOR** Information change or deletion by a internal or external agent. Harm could be done to many Patients, and this could be very harmful.
3. **LIKELY*MODERATE** Information being stolen by a virus. It is more likely that patient data could be the focus of a virus.

ConceptService.java

Type of Asset: Software

Class: PHI

Value of Asset: Major

Using the methods in this class, it is possible to do a lot with Concepts, Drugs, Concept Proposals, including deleting and updating for instance.

Threat Agents:

- Intentional internal agent.
- Intentional external agent.

- Accidental internal agent.

Threats:

1. **POSSIBLE*MINOR** Internal or external intentional agent that wants to steal or change information about Concepts.
2. **POSSIBLE*MINOR** Internal agent accidentally change Concepts or Drugs definitions.

AdministrationService.java

Type of Asset: Software

Class: Other

Value of Asset: Major

This asset enables some administrative tasks to be done in the system.

Threat Agents:

- Intentional internal agent.
- Intentional external agent.
- Accidental internal agent.
- Virus

Threats:

1. **LIKELY*MINOR** Internal intentional agent that wants to get unauthorized information.
2. **POSSIBLE*MODERATE** External agent that wants to change information, accessing the sets or removes; for example, removing global properties.
3. **POSSIBLE*MODERATE** Virus programmed to change information.

Risks

Extreme Risks

Context.java - 1

This threat concerns an insider or hacker gaining superuser rights to the database or services.

For authentication, the only verification made concerns verifying if it is a Daemon Thread (used for startup and scheduling), what would generate a error saying that authenticating is not necessary or allowed.

There are mechanisms for auditing logging attempts to authenticate, but they are used just for debugging. Enabling it for default run should be considered for keeping track of possible malicious attempts to authenticate.

My understanding of how this class work is good, besides being a pretty big piece of code, but I may be missing some mechanisms implemented in other places of the system. Therefore, the assessment was made thinking locally.

The authentication at this layer is pretty simple because it is not really implemented here, but in a lower layer (at UserContext and ContextDAO). Anyway, it is important to reinforce security in this layer so there is more chance to avoid an attack.

PatientService.java - 2

The main control OpenMRS has to deal with such unauthorized access in this class is the @Authorized annotation. This annotation refers to another class that deals with security; the name of the class is AuthorizedAnnotationAttributes, and it is imported as org.openmrs.annotation.Authorized in the beginning of the file. In order for one to change or delete information, it has to pass through this authorization process. This can be seen in the function "public Patient voidPatient", where "@Authorized({ PrivilegeConstants.DELETE_PATIENTS })" is right before the function's code starts, and in every function that changes some patient information, as public void updatePatientIdentifier that has the annotation "@Authorized({ PrivilegeConstants.EDIT_PATIENT_IDENTIFIERS })".

Another control that I could identify is the fact that when a patient is deleted through a expected way, it is possible to reverse the process. There is the function "public Patient unvoidPatient", which will, as they comment in the code, "revive" a Patient. What happens is that when the function to delete a Patient is called, the patient is voided, which means that they are still in the database but will not appear to the end user. The function to void a Patient requires the same privileges as the function to unvoid one. If a patient, nevertheless, is deleted using the "public void purgePatient", it is deleted from the database and the processes seems to be irreversible. In order to use the function to purge a patient, other privileges are required, which is good.

I am confident about my assessment because I looked throughout the code and made researches to understand it. I may have missed something, for OpenMRS is a very large project, and this specific assessment involves more than only this asset; the purge may have more treatment in the database level and I am not aware of it. My answer, however, explain how that threat is addressed at this class, and I believe it is complete.

My general impression of the controls is that they are good. Checking for authorization in every one of those requests is the first thing in this situation, and the possibility of reversing the deletion is another very important thing. I feel that this code is well made to support this methods.

PersonService.java - 2

The main control OpenMRS has to deal with unauthorized access in this class is the @Authorized annotation. This annotation refers to another class that deals with security; it is imported as org.openmrs.annotation.Authorized in the beginning of the file. In order for one to change or delete information, it has to pass through this authorization process. This can be seen

in the function "public Person savePerson", where "@Authorized({ PrivilegeConstants.ADD_PERSONS, PrivilegeConstants.EDIT_PERSONS })" is right before the function's code starts, and in every function that changes some patient information, as "public PersonName savePersonName" that has the annotation "@Authorized({ PrivilegeConstants.EDIT_PERSONS })" for instance.

Another control that I could identify is the fact that when a person is deleted sing the function "public Person voidPerson", it is possible to reverse the process. There is the function "public Patient unvoidPerson", which will, as they comment in the code, "revive" a Person. What happens is that when the function to delete a Person is called, the patient is voided, which means that they are still in the database but will not appear to the end user. The function to void a Person requires the same privileges as the function to unvoid one. If a person, nevertheless, is deleted using the "public void purgePerson", it is purged from the database and this process seems to be irreversible. In order to use the function to purge a person, other privileges are required, which is good.

I am confident about my assessment because I looked throughout the code and made researches to understand it. I may have missed something, for OpenMRS is a very large project, and this specific assessment involves more than only this asset. My answer, however, explain how that threat is addressed at this class, and I believe it is complete.

My general impression of the controls is that they are good. Checking for authorization in every one of those requests is the first thing in this situation, and the possibility of reversing the deletion is another very important thing. I feel that this code is well made to support this methods.

UserContext.java - 1

This threat concerns an insider or hacker gaining superuser rights to the database or services.

There are no security mechanisms implemented. The attempt to authenticate just fails by throwing an Exception from a lower layer (database checking).

There are mechanisms for auditing logging attempts to authenticate, but they are used just for debugging. Enabling by for default run should be considered for keeping track of possible malicious attempts to authenticate.

Understanding this part of the code is not difficult, but understanding the whole piece is. Therefore, the assessment was made thinking locally.

The authentication at this layer is pretty simple because it is not really implemented here, but mainly at the database layer. Anyway, it is important to reinforce security in this layer so there is more chance to avoid an attack.

UserContext.java - 3

This threat concerns an insider or hacker getting sensitive information about the user.

There is no control about the access to the User object wrapped by this class. If authenticated, the access is opened to all information, never asking the password again or checking for the authenticity of the user.

When sensitive information is accessed, assuring in someway that the authentication is still valid is a way of increasing security.

My understanding of how this class work is good, besides being a pretty big piece of code, but I may be missing some mechanisms implemented in other places of the system. Therefore, the assessment was made thinking locally.

There is not a mechanism for avoiding this kind of threat but there are logs made for auditing, what can be used for detecting the threat.

UserContext.java - 4

There is no security mechanism implemented at this layer concerning privileges needed for altering User roles. There is a check if the user is a SuperUser to allow it to become other user but nothing else.

There are also mechanisms for auditing attempts to become other user and to change user information, but they are used just for debugging. Enabling it by default run should be considered for keeping track of possible malicious attempts to use this function.

Understanding this part of the code is not difficult, but understanding the whole piece is. Therefore, the assessment was made thinking locally.

Security about roles assignment and privileges of an user need to be reviews. It is important to reinforce security in this layer so there is more chance to avoid an attack.

High Risks

ActiveListServiceImpl.java - 1

This threat concerns an insider or hacker to add an improper service to a session..

There is no security mechanism implemented at this layer, even though there is a comment alerting that this class should not be used by its own.

My understanding of how this specific class work is good, but it works together with some other parts of the system, so it is a little difficult to assess it.

At a first view, this class permits direct manipulation of the list of services being offered by the system. It can result in integrity problems as well as availability problems, once an attacker can cause the system to break down.

AdministrationService - 2

There is the "@Authorized(PrivilegeConstants.MANAGE_GLOBAL_PROPERTIES)" that checks the privilege of one trying to save a global property using the function "public GlobalProperty saveGlobalProperty", but when one would try to set, update or remove a global property by other means such as using "public void setGlobalProperty", there is no pivilege check.

The function "public void setGlobalProperty" does the same as "public GlobalProperty saveGlobalProperty", but the first one does not check the privileges. It seems an obvious control that is missing - put the @Authorized annotation before "public void setGlobalProperty", or define this as "deprecated".

The @Authorized annotation is also missing in the functions "public void updateGlobalProperty", "public void addGlobalPropertyListener", and "public void removeGlobalPropertyListener".

I am confident about my assessment. I was looking for the @Authorized annotations and could find it in some functions, but in some other ones, listed above, it is not present.

My impression is that OpenMRS controls this threat not too well. It controls only the basics.

AdministrationService - 3

There is the "@Authorized(PrivilegeConstants.MANAGE_GLOBAL_PROPERTIES)" that checks the privilege of one trying to save a global property using the function "public GlobalProperty saveGlobalProperty", but when one would try to set, update or remove a global property by other means such as using "public void setGlobalProperty", there is no privilege check.

The function "public void setGlobalProperty" does the same as "public GlobalProperty saveGlobalProperty", but the first one does not check the privileges. It seems an obvious control that is missing - to put the @Authorized annotation before "public void setGlobalProperty", or define this as "deprecated".

The @Authorized annotation is also missing in the functions "public void updateGlobalProperty", "public void addGlobalPropertyListener", and "public void removeGlobalPropertyListener".

I am confident about my assessment. I was looking for the @Authorized annotations and could find it in some functions, but in some other ones, listed above, it is not present.

My impression is that OpenMRS controls this threat not too well. It controls only the basics.

Context.java - 2

This threat concerns an insider or hacker trespassing the system by becoming other user.

There is no security mechanism implemented at this layer, but there is a comment alerting that you should only be able to do this as a superuser.

There are also mechanisms for auditing attempts to become other user, but they are used just for debugging. Enabling it for default run should be considered for keeping track of possible malicious attempts to use this function.

My understanding of how this class work is good, besides being a pretty big piece of code, but I may be missing some mechanisms implemented in other places of the system. Therefore, the assessment was made thinking locally.

This layer has some one part of the procedure for becoming another user (change locale), but no security is implemented here. It is important to reinforce security in this layer so there is more chance to avoid an attack.

Context.java - 3

This threat concerns an insider or hacker forging sessions to get inappropriate rights.

The system keeps track of all sessions opened and closed. There is a comment note saying that all "units of work" should be surrounded by openSession and closeSession calls. Sessions opened with the current user are not traced.

Some sort of check about a high number of sessions with high privileges (or from the same user) would be useful.

My understanding of how this class work is good, besides being a pretty big piece of code, but I may be missing some mechanisms implemented in other places of the system. Therefore, the assessment was made thinking locally.

There is not a mechanism for avoiding this kind of threat but there are logs made for auditing, what can be used for detecting the threat.

Daemon.java - 1

This threat concerns an insider or hacker trying to run tasks and sessions with elevated privileges.

There is not any sort of privilege check or security mechanism for executing tasks, but exploring this feature would depend on how tasks are assigned to the scheduler.

My understanding of how this specific class work is good, but it works together with some other parts of the system, so it is a little difficult to assess it.

It has a very well defined function in the system and it is somewhat difficult to exploit it. There is not much place for auditing neither consistent security mechanisms, but it would be hard to think about a way of both attack it or secure it.

PatientService.java - 1

There are two main controls here. The first one is the check of the authorization. As noted before, in order for someone to change or delete a patient, they have to have certain privileges, which are evaluated by the @Authorized annotation. The annotation is present and it seems good. The other main control is the possibility to reverse the deleted Patient. This is very important in this case because the unintentional deletion can happen when information is needed fast, so a manner of undoing the action is desirable. I also found that deleting a Patient identifier can be undone, or it can be purged as well and this action won't be undone.

If the action can't be undone, I surmise that OpenMRS doesn't require any other permissions in the database to purge a Patient or a Patient identifier. It should be some type of control that would test privileges again in the database level, and this way slowing down an attack.

I am confident about this assessment since it looks a little like the another one, and in this case, the process is more reversible because the accidental threat is less likely to cause much data loss, implying in an at most moderate average case threat.

OpenMRS seems to be, in my general impression, correctly concerned about the authorization and authentication. This threat seems to be well handled in the average cases it could happen.

PatientService.java - 3

The main control here is still the @Authorization annotation, but this time the authorization is focused in the get functions. There are some functions found to get Patient information: 3 different public List getPatients, for different types of searches; 1 public Patient getPatientByExample; 1 public List getDuplicatePatientsByAttributes; 2 public Integer getCountOfPatients. There are also functions to get allergies and problems of a patient, which are PHI. All the functions have the @Authorization annotation as a control.

I think there should be a different control for some different kinds of searches. For instance, the @Authorization annotation used to check the privileges to get patients is the same in all the functions (@Authorized({ PrivilegeConstants.VIEW_PATIENTS })). So, if a virus has access to one, it has access to everything, even the voided Patients.

I am confident in this assessment, since the code has been well studied by myself. I was looking for auth controls with regard to this threat, I think they are summarized here well and my thought about a missing control can be relevant.

My general impression of how OpenMRS deals with this treated is that they are in a more basic level, using the same method of authorization to every get of a similar type, when there should be a more complex way of treating this matter.

PersonService.java - 1

There are two main controls here. The first one is the check of the authorization. As noted before, in order for someone to change or delete a person, they have to have certain privileges, which are evaluated by the @Authorized annotation. The annotation is present and it seems good. The other main control is the possibility to reverse the deleted Person. This is very important in this case because the unintentional deletion can happen when information is needed fast, so a manner of undoing the action is desirable. I also found that deleting a Person attribute can be undone, or it can be purged as well, and this last action won't be undone.

I am confident about this assessment since it looks a little like the another one; in this case, the accidental threat is less likely to cause much data loss, implying in an at most moderate average case threat.

OpenMRS seems to be, in my general impression, correctly concerned about the authorization and authentication. This threat seems to be well addressed.

PersonService.java - 3

The main control here is still the @Authorization annotation, but this time the authorization is focused in the get functions. There are some functions found to get Person information, including "public Set getSimilarPeople" and "public List getPeople" for example. There are also functions to get a Person's relationships and attributes.

I think there should be a different control for some different kinds of searches. For instance, the @Authorization annotation used to check the privileges to get people is the same in all the functions (@Authorized({ PrivilegeConstants.VIEW_PERSON })). So, if a virus has access to one, it has access to everything, even the voided Persons.

I am confident in this assessment. I was looking for authorization and authentication controls with regard to this threat, I think they are summarized here well and my thought about a missing control can be relevant.

My general impression of how OpenMRS deals with this treated is that they are in a basic level, using the same method of authorization to every get of a similar type, when there should be a more complex way of treating this matter.

Service implementations - 1

This threat concerns an insider or hacker using or modifying a service improperly.

There is no security mechanism implemented for this thread. Nothing assures the correct usage of the services by an user. Also, the system does not keep much trace of what happens in it.

There is not any sort of privilege check for assuring correct usage, passing the impression that any user can do everything.

There are a lot of classes involved in this threat, so it is impossible to check for every line of code in them to assure that our assessment is correct. By the way, our understanding of the function of this classes is fine.

In this case, it is important to implement some sort of auditing and add checkers to important services offered by OpenMRS, which are visibly lacking.

Service implementations - 2

This threat concerns an insider or hacker getting sensitive information from services.

There is no privilege checking in the service implementations, neither the auditing/logging mechanisms found at other pieces of code.

Sensitive information must be protected by privilege checking or any sort of restrictions so that the system assure security instead of relying on the user.

There are a lot of classes involved in this threat, so it is impossible to check for every line of code in them to assure that our assessment is correct. By the way, our understanding of the function of this classes is fine.

It is an important to assure that the access to sensitive information is wrapped in some way by a security mechanism that is lacking in OpenMRS.

UserContext.java - 2

This threat concerns an insider or hacker providing false information about an user.

There is no security mechanism implemented at this layer concerning privileges needed for altering user information. Some protection of the access to an user information is visibly lacking.

There are mechanisms for auditing attempts to become other user and to change user information, but they are used just for debugging. Enabling it by default run should be considered for keeping track of possible malicious attempts to use this function.

Understanding this part of the code is not difficult, but understanding the whole piece is. Therefore, the assessment was made thinking locally.

It is important to reinforce security in this layer so there is more chance to avoid an attack.

Medium Risks

AdministrationService - 1

There is the "@Authorized(PrivilegeConstants.VIEW_GLOBAL_PROPERTIES)" that checks the privilege of one trying to get all the global properties, but when one would try to get a global property by other means such as using "public List getGlobalPropertiesByPrefix", there is no pivilege check.

The controls for checking the privileges are missing in this class. There should be controls (@Authorized annotations) for the functions "public String getGlobalProperty", "public String getGlobalProperty", "public GlobalProperty getGlobalPropertyObject", "public List getGlobalPropertiesByPrefix", "public List getGlobalPropertiesBySuffix" and "public Set getPresentationLocales".

I am confident about my assessment. I was looking for the @Authorized annotations and could find it in some functions, but in some other ones, listed above, it is not present.

My impression is that it does not seem that OpenMRS has an adequate control for this threat. It controls only the basic; that is, it checks when someone wants to get all global properties, or administrative services, but it doesn't check for other important things.

ConceptService.java - 1

The control OpenMRS has to deal with unauthorized access in this class is the @Authorized annotation. This annotation refers to another class that deals with security; it is imported as org.openmrs.annotation.Authorized in the beginning of the file. In order for one to get or change information, it has to pass through this authorization process. This can be seen in the function "public Concept getConceptByUuid", where "@Authorized(PrivilegeConstants.GET_CONCEPTS)" is right before the function's code starts, and in every function that changes some patient information, as public Drug saveDrug" that has the annotation "@Authorized({ PrivilegeConstants.MANAGE_CONCEPTS })" for instance.

I am confident about my assessment because I looked throughout the code and made researches to understand it. My answer explains how that threat is addressed at this class, and I believe it is complete.

This thread seems to be very well addressed. There is an authorization checking for every method that needs it. Moreover, like the PatientService.java and PersonService.java, it is possible to undo an action of deleting a Concept or a Drug, and this is also under authorization checking.

ConceptService.java - 2

The control OpenMRS has to deal with unauthorized access in this class is the @Authorized annotation. This control is the same the threat "ConceptService.java - 1", since they can be

addressed equally. This annotation refers to another class that deals with security; it is imported as org.openmrs.annotation.Authorized in the beginning of the file. In order for one to get or change information, it has to pass through this authorization process. This can be seen in the function "public Concept getConceptByUuid", where "@Authorized(PrivilegeConstants.GET_CONCEPTS)" is right before the function's code starts, and in every function that changes some patient information, as public Drug saveDrug" that has the annotation "@Authorized({ PrivilegeConstants.MANAGE_CONCEPTS })" for instance.

I am confident about my assessment because I looked throughout the code and made researches to understand it. My answer explains how that threat is addressed at this class, and I believe it is complete.

This thread seems to be very well addressed. There is an authorization checking for every method that needs it. Moreover, like the PatientService.java and PersonService.java, it is possible to undo an action of deleting a Concept or a Drug, and this is also under authorization checking.

Context.java - 4

This threat concerns an insider or hacker trying to shutdown the system from the outside.

The system keeps track of the process of startup and shutdown for debugging purposes, what can be useful for security.

There is not any sort of privilege check for shutdown, any user can do it. Some sort prompt for confirmation by the user or error in case of activities running would prevent malicious use of this function.

My understanding of how this class work is good, besides being a pretty big piece of code, but I may be missing some mechanisms implemented in other places of the system. Therefore, the assessment was made thinking locally.

It is not a critical function and creating security mechanisms for it can be conflict with some use cases or requirements of the system.

Design Principles

Economy of Mechanism

Keep authentication as simple as possible, assuring security without making it too complex to be used or too elaborated so that the risk of a failure is imminent.

B There is not too many mechanisms implemented for securing authentication, but the authentication system itself is quite secure. It has a role-based authentication system that is certainly a good point and does not make it too complex.

Open Design

The authentication system does not rely on secrecy of its implementation.

A The authentication system is well documented and open for all the community, so it does not rely on secrecy.

Least Privilege

Each user should authenticate to perform only the necessary tasks for her/his role.

A- What can be done in the API level is fairly well done because most of the methods check for specific privileges before running. There are some methods, though, that do not perform this operation, and therefore every user could run them even if it isn't part of their tasks.

Least Common Mechanism

There shouldn't be various ways of authenticate to perform the same task.

B If you have access to the Person class and can use the methods, you can also do a great change to Patients, which extends the Person class. There is, however, a separate privilege to work only with Patients. If an attacker can hack the Person method, Patient is not secure; so to perform changes in a Patient, the only mechanism should be through the Patient class. If it is desired that a person who can change Persons should also be able to change Patients, then this person should be assigned with the privilege of changing Patients specifically and separately.

Psychological Acceptability

The authentication is made where necessary and should not inhibit users from doing what they know they should do.

A The privileges are well separated, but not more than necessary. The user who can use getConcept can also use getConceptByUuid for instance, which makes sense and needs the same authentications. I feel that the users will not consider that there are unnecessary checks for authentication.

Layering

Authentication should be implemented and checked at the API even if it was already checked somehow before; and also should checked at other parts of the OpenMRS.

A- The OpenMRS API checks for authentication fairly well, and if the the other parts of the system also do, it is well separated in layers. There is, however, the possibility of purging a Person from the database through the API, and I am not sure if the authorization to do so is checked again at the database; it is not our part of the project.

Summary of Findings

The main problem with the authentication and authorization system at OpenMRS is the lack of verification of privileges in some places of the system that can generate attacks to sensitive information or availability. It hurts the Complete Mediation principle concerning the correct authorization for access to sensitive information.

There is also some violation of the design principle of Isolation - it is possible to delete a Patient through two different privileges and two different methods. This increases the risks to a harm in a Patient implementation.

Recommendations

Keep track of authentication attempts

Detecting attack attempts is the first step for security. In case of successive wrong attempts to authenticate, it is clear that there is something wrong. Keep track of attempts is a way of detecting malicious users and then banning them.

OpenMRS already have some debugging messages implemented and turning them into an audit system is a simple solution. The creation of a service specific for it storing attempts at a database is a more elaborated solution.

Establish some limit for faulty authentication attempts

Dictionary attacks are one of the main threats to authentication systems. Delaying or avoiding a great amount of attempts is a simple way to avoid it.

One solution is to establish a minimum time for authentication attempts. Setting this time to 1 second would limit the number of attempts per day to less than 100,000. This, for a dictionary attack, is not a significant amount of attempts.

Something more smart and secure is to limit the number of wrong attempts. For example, limit it to three and at the fourth faulty attempt ask for the user to change their password.

Establish password reinforcement policies

This also refers to dictionary attacks but it is also a way to increase authentication security as a whole. Force the users to have a strength password is simple but it reduces the relative usability of the system (difficult passwords are hard to remember).

Checking for the strength of a password is easy. Regular expressions can be used to assure that some simple rules are followed, for example force it to have numbers and leters or special characters.

Also, some suggestions of how to make users create and remember of the password is a good way to make users accept it.

Enforce installation rules with the intent of keeping OpenMRS binaries secure

OpenMRS is a big project with several parts, layers, and packages that generate a great amount of binary files (.class) and database data. For security reason, the access for edition or deletion of these files must be controlled.

It can be done by the administrator of the system where OpenMRS is installed by restricting access to some directories. In case of UNIX system, commands like chmod and chown could do the job. Windows and Mac OS systems also have similar commands.

The permission modifications may assure that just the root user (superuser) or the owner of the files can have access to them. When a new compilation is needed, just these user will have sufficient rights for placing the binaries where it is supposed to run. A development environment also will be needed for this reason.

Conclusion

OpenMRS is a system that in a first view does not offers so much place for worries, once it does not deal with a great amount of sensitive information like financial information, and the main purpose of the system is scientific. However, it is true that privacy is a big concern in a system like OpenMRS, once it have a great amount of information about patients and clinical conditions. This should not mean that OpenMRS is not a place for big attacks, but it means that it can give the impression that security is not needed, when the truth is that it still have place for improvement.

OpenMRS API Authentication/Authorization May, 2016

Authors: Trevan A., Sergio A., Marisa A., Seung Ho R.

Executive Summary

OpenMRS is a great open-source program that enables to record a patient's condition electronically; however, through this assessment, we found some security vulnerabilities that need a developer's attention right now. The team especially researched about the authorization and authentication process on API (Application Programming Interface) layer. OpenMRS has some important assets that should not be leaked by attackers. For example, patient information is a critical data value related to PHI (Protected Health Information). This one includes a patient's name, place of living, dieases, current medical condition and so on. This kind of information is a good target for organized criminals (outside attackers) because it may include a patient's credit card numbers and social security numbers. Also, employees (inside attackers) can leak this sensitive data to public accidentally or deliberately.

The Risks in OpenMRS can be divided by in four categories: extreme risks, high risks, medium risks and low risks. Hackers and SQL injection attacks are in extreme risks; they rarely happen in real conditions, but they cause serious aftermath when they actualize. Malicious employees and social engineering hackers come under high risks.

For the last point, the team evaluated that OpenMRS follows each design principle or not. Design principles are important to reinforce security mechanisms of a program and help to determine a program is developed by basic security rules. The details of design principles in OpenMRS is described in the Summary of Findings in below.

Scope

This is a wiki page that refers to the the Authentication/Authorization Application Programing Interface(API) of the OpenMRS software. This API is meant to handle the authentication of users and their passwords to OpenMRS. The API also handles which users will have authorization to perform particular actions within the OpenMRS system. For example some user(such as a doctor) may have access to a particular individuals data because that individual is a patient of that user. Other users who do not have a particular relation to a specific patient will not have access to that users data. We will discuss the features/problems related to the Authorization API.

A patient's name is one of the important aspects that configure electronic medical record system, enabling to identify a patient's diseases and medication. However, many programmers and medical teams overlook to protect the information because they assume that hackers cannot do anything even though hackers can access only to one's name. There is a file which is called IdentifierValidator.java

under the directory openmrs-core-master/api/src/main/java/org/openmrs/patient. In this source code file, it takes a patient's name by public String getName();. The identifier checks the parameter which is one's name and return it to the validator if it is correct; however, if it is not correct, it throws the name to UnallowedIdentiferException. We found a potential vulnerability (problem) in here: the access modifer, “public.” In Java language, there are four access modifiers which control access from other classes, packages, subclasses and worlds: public, protected, no modifier and private. The public modifier allows all the accesses from a class, package, subclass and world. It makes easy when a program has to import from other programs (like in this situation), but it also means another class or package can access to “String getName()” without any barriers (security protections that prevent numerous attacks from hackers). A patient's name is also an important asset that should be protected on all the layers and can be accessed by limited people who gets authorization; therefore, we consider that it can be one of the potential problems that are related to OpenMRS API Authorization.

In the OpenMRS Directory there is a folder labeled validator that contains a large amount of java files used for validation. This is interesting because all these files are used to check data integrity and make sure users that are attempting to input or edit data in the system must be authorized to do so. Some of these are very useful for keeping private and personal patient data secure, and they use classes such as ValidateUtil that contain useful methods like validateFieldLengths to make sure attackers cannot insert data that would break the program. There is, however; a method in ValidateUtil that can be called named disableValidation which would void all enabled validations placed to protect data.

In the OpenMRS Directory, there is code called UserContext.java. This java code is very interesting because it deals with direct authentication. UserContext.java authenticates a single user with the appropriate privileges, permissions, and roles. UserContext.java also allows for refreshing the user, changing current authentication to become another user, store location, and obtain all the roles of the current user. UserContext.java imports authentication exceptions so as to deal with unauthorized users or users trying to access certain parts of the system they don’t have the privileges for.

Developer Notes:
/openmrs-core-master/api/src/main/java/org/openmrs/patient/IdentifierValidator.java
openmrs-core-master/api/src/test/java/org/openmrs/RoleTest.java
openmrs-core/api/src/main/java/org/openmrs/api/context/UserContext.java

Assets

Patient Information

Type of Asset: Data

Class: PHI

Value: Critical

Patient information is all the data in the database about the clients of clinic using OpenMRS. This information ranges from a patients name and place of living to all of their medical information and health complications that have been recorded. This information should only be accessed by a patient's doctor and therefore authorization is extremely important to this asset.

Threat Agents:
• Organized Criminals: Patient information can include credit card numbers and social security numbers. Criminals looking to sell identity information would be viable threat agents to attack medical record software to access patient information.
• Employees: Anyone that might have bad blood with a patient has the potential to leak information, or be subjected to social engineering to mistakenly grant access

Threats:
• Glitch: unlikely*minor: possible glitch in the program could result in a loss of data and availability that would cause many problems
• Being Hacked: likely*major: if a threat agent was really interested in patient information it may be able to gain access by finding a weakness or gaining false authentication through social engineering.

Usernames and Passwords of the Authorization System

Type of Asset: Data

Class: SEC

Value of Asset: Critical

Usernames and passwords of the authorization system are keys of consisting OpenMRS. Medical teams and researchers who frequently use OpenMRS can access to a patient's data by this authentication system. Therefore, if the information is hacked or leaked by a hacker or a system administrator, then most of the data can be abused.

Threat Agents: Hackers (outside attackers) and system administrators (inside attackers)

Threats

Unlikely Catastrophic: It can be hard to access encrypted databases which contain the usernames and passwords of the system (it is much harder to get both of them). However, once hackers succeed to access the usernames and passwords data, then they easily look around other sensitive information that is related to one patient's medical record.

Likely Major: System administrators usually take care of the authorization and authentication of the system. For this reason, there is a high possibility that a system manager can access and capture all the information of usernames and passwords. However, a lot of modern systems record a user's activity by log files or others, so a system administrator's suspicious activity can be easily detected by other IT professionals.

Authorization Database

Type of Asset: Data

Class: SEC

Value of Asset: Major

The database which holds usernames and passwords is an essential and important part of the the authorization API. It is important that the information in this database be encrypted using salt and hash techniques to prevent possible threats from attaining this information.

Threat Agents:
Employees
Hackers

Threats:
Unlikely/Major:An Employee could be a threat in the senario where he/she would have access to the database and may have malicious intent. With access to the database they could remove or gain database information which would be critical to a successful operation.

Likely/Major An outsider Threat or hacker could possibly do SQL injections to gain access to the database. They might also be able to gain access through back doors created by software developers. With access they could possibly gain authorization information or do high damage to data integrity.

Permissions/Roles

Type of Asset: Data, communication

Class: Other

Value of Asset: Major

Permissions are a really important asset. With Permissions, an individual only has access to a certain part of the system. The more permissions they have, the more access they have to the overall systems databases, data, permissions of others, functionality and so on. The best practice to prevent someone from seeing and having too much access is separation of privileges. Even if one part of the system is compromised, the whole system remains intact.

Threat Agents:
- Employee
- Hacker

Threats:
• The Employee is one threat agent. The employee already has permission(s) to some part of the system. There are two things that can happen. This employee can compromise whatever part of the system they have access to, no matter how small. This employee can try to obtain the other permissions to gain access to the rest of the system. The subjective likelihood that this threat will happen is low but the but the seriousness if the threat does happen can range from low to doomsday depending on how many permissions the employ can get.
• The hacker is one threat agent. The hacker will attempt to employ either code based attacks or social engineering to obtain access and gain permissions. Through social engineering, they'll most likely using phishing attempts to ascertain data from other employees. Through code based attacks, they will attempt SQL injections, look for code vulnerabilities, or unused code. The subjective likelihood that this threat will happen is low but the but the seriousness if the threat does happen can range from medium to doomsday depending on how successful the phishing or code based attack is.

Design Principles

Economy of Mechanism

Economy of mechanism is that practice of keeping things simplified as to avoid too much complexity. This allows for less to go wrong in the event of a problem, as well as easier troubleshooting since there's less to look at.

C The system in openMRS is rather complex, though it seems like a difficult thing to simplify. It would be useful if the files were more organized in terms of which files dealt with which API. The source code layout could also be put in a more simplified manner as to make it easier to navigate.

Open Design

From the principle of Open Design, all security mechanisms in OpenMRS should be open to public including all developers who help this open source project. The source code of OpenMRS is opened to everyone who are interested in the project and security mechanisms are readily understood by the programmers.

A It highly follows the rules of Open Design, providing all of security mechanisms of the program very clearly. Some of the security policies in OpenMRS may not be satisfy authorization, authentication, privacy and other security-related laws in developed countries, but it generally does not cause a significant problem in developing nations which highly need an electronic medical record system such as OpenMRS.

Separation of Privilege

The system is weak to keeping sensitive information confidentially. As we found and assumed some security weaknesses above, most of the information can be transferred through public class and implication files. Another source code program can call a patient's important data that should

not be revealed to others without a problem. So, a system administrator can access to one patient's medical record even though it is not necessary to consist or repair a system or database because, in this algorithm, one person can enter private information easily.

E Especially, whole of a patient's medical record should be encrypted and cannot be accessed by a developer, system administrator or other IT professionals - only a medical team or the people who get an approval from a physician should enter sensitive information of a patient.

Psychological Acceptability

Psychological Acceptability means that security shouldn't be a hindrance to the user. At the same time, security should also be able to meet the needs of those with authorized access. From the view point of the user, it seems the security isn't a hindrance. When it comes to authenticating the user, all security is done in the code to authenticate the user. There doesn't seem to be anything annoying like having the user prove he is not a robot or locking the user out if they get there password wrong more than once. This also seems to hold true when the user enters his data that identifies them as a patient.

A. It seems to follow Psychological Acceptability very closely. Security doesn't get in the way of the user.

Isolation

There are three parts to isolation. 1.) Public access systems should be isolated form critical resources. 2.) The processes and files of individual users should be isolated from one another except where it is explicitly desired. 3.) Security mechanisms should be isolated in the sense of preventing access to those mechanisms.
In terms of 1, this seems to be more of a front end aspect. Judging from the front end user interface, there seems to be no way to access critical resources like the data base or security protocol. However in the background, you can access information from critical resources or rather parts of it since a lot of the getters to this information are public instead of private.
In terms of 2, UserService.java seems to incorporate this principle. It ensures that processes and files are separated based on authenticating the permissions of the current logged in user. Beyond this, I'm not too sure.
In terms of 3, OpenMRS also seems to follow this. All the files related to these security mechanisms are all header files. It seems that the compilations files are all hidden. There is no direct way of accessing or seeing these security mechanisms.

Overall Grade: A

Encapsulation

The design principle Encapsulation is used to in object oriented programming to bind data together, allowing programmers to protect data from outside interference. It is important because it allows the hiding of data.

A In the OpenMRS source code was well encapsulated by the use of controlling data using protected, private, and public classes. Well through the API code there is ample evidence of the use of these classes which allow for excellent use of encapsulation.

Layering

The layering principles deals with securing each layer of of the software so that if there is a breach in one section then the other others will still be safe and secure. For example, patient data and information would be separately secured from user name and password information. That way if one of these sections of the database was breached the other would not be compromised.

"B' OpenMRS uses the hasPrivilege() function to give groups access the layers with different levels of security. This is effective for keeping lower level users out of places they do not belong, but if a hacker were able to change which group of users they belonged to they would gain access to information they should not have.

Recommendations

- According to the separation of privilege, a program should be divided in several parts that are limited to specific privileges to perform a task. This design principle enhances security levels because hackers are unable to control whole parts of a system; each process or user needs specific privileges to handle works of a program. However, in OpenMRS, the system is weak to keeping sensitive data confidentially and it does not classify each privilege or permission clearly. Most of the programs in OpenMRS can easily call other classes and implication files without any barriers. Also, there is no rule or standard to limit each user's specific role or privilege. It is possible that an authorized user can access to a patient's medical record which should be highly encrypted and only accessed by limited physicians, not a user, program developer, system administrator or other IT professionals.

Conclusion

OpenMRS is an extremely useful tool that allows for electronic patient records where they aren't as widely available. This allows for developing countries to have an excellent way of storing patient data where they otherwise wouldn't be able to. Through the availability of open source software, many developers are able to work together to create an efficient, and useful product which can be used around the world. With more emphasis on security in future versions, OpenMRS will be able to better protect the personal data of its users. It will also also allow for OpenMRS to expand into more countries where privacy laws are greatly stressed.

While OpenMRS is an extremely useful tool, it needs a lot of work. OpenMRS tries to faithfully follow the design principle of being open, but we have found that its compilation files are hidden. There are a lot of header files like serialization and password exceptions that get called and work but we don't know how they work since the compilation files are nowhere to be seen. There are no barriers to accessing data that you are in turn makes assigning permissions through the java programs like role services. The source code is disorganized and quite cumbersome to navigate. There isn't enough categorizing to easily identify related files and the lack of

commenting doesn’t help. The poor documentation of comments doesn’t explain what is going on, what it is being used of, and why it has to be used through extensions. With some of these issues being solved, it will greatly improve the development of OpenMRS.

API Accountability/Auditing May, 2015

Scope

Our group will be assessing the API layer of OpenMRS. The API allows developers to interact with the OpenMRS data model. The API layer deals with services, logic, and the database handler. Our group's assessment will be focused on the auditing of the database handler and logic portion of the API.

This assessment will be reviewing how the system currently audits the access and change patients' health information. We will also be suggesting what information the system should audit, that it currently does not. We will be assessing the database handler to make sure it is secure against SQL injection attacks. It is very important for the confidentiality of the client that this layer is secure. If this layer is secure, then the information inside the database is secure.

To install the OpenMRS API source code we did the git clone command in the console of our computer like so **git clone** https://github.com/openmrs/openmrs-core . On the first attempt the clone stopped at 57% an attempt the reclone was done. On the second attempt the code fully downloaded after a couple minutes of downloading the entire API source code was in the directory.

Assets

PersonName.Java

Type of Asset: Software

Class: PII

Value of Asset: Minor

The asset contains the data entry point for a patient's name information.

Threat Agents:
Hackers looking for name personal information
accidental insider leak

Threats:Possible*Minor

PersonAddress.Java

Type of Asset: Software

Class: PII

Value of Asset: Minor

The asset contains the data entry point for a patient's address information.

Threat Agents:
Hackers looking for name personal information
Stalkers
accidental insider leak

Threats:Possible*Moderate

Drug.Java

Type of Asset: Data

Class: PHI

Value of Asset: Moderate

The asset contains the data entry point for a patient's medication type and dosage information.

Threat Agents:
Potential Thieves looking for certain medications
accidental insider leak
insiders looking to steal drugs from patients

Threats:Likely*Moderate

OrderUtil.Java

Type of Asset: Data

Class: Other

Value of Asset: Moderate

The asset contains the order form for supplies.

Threat Agents:
Hackers looking to place orders for themselves
Insiders looking to steal from the company

Threats:Possible*Moderate

DrugOrder.Java

Type of Asset: Data

Class: Other

Value of Asset: Moderate

The asset contains the drug order form for patients.

Threat Agents:
Insiders looking to steal and sell drugs
Hackers looking to order drugs

Threats:Likely*Moderate

User.Java

Type of Asset: Data

Class: SEC

Value of Asset:Major

The asset consists of the ability to change user role and permissions.

Threat Agents:
Hackers looking to extort bribes for permissions
Vandals
Insider accident

Threats:Possible*Major

PatientDAO.Java

Type of Asset: Data

Class: SEC

Value of Asset: Major

The asset contains the ability to delete users from the system.

Threat Agents:

Hackers trying to disrupt the system or remove patient's information

Threats:Possible*Moderate

Order.Java

Type of Asset: Data

Class: PHI

Value of Asset: Major

The asset consists of the ability to change a patient’s treatment or give patient new medications.

Threat Agents:

Patient looking to order unneeded medication

Insider trying to order extra medication

Threats: Possible*Moderate

Allergy.Java

Type of Asset: Data

Class: PHI

Value of Asset: Minor

The asset keeps track of patient’s allergies.

Threat Agents:

Hacker looking for weakness of a patient

Threats: Possible*Severe

AllergySeverity.Java

Type of Asset: Data

Class: PHI

Value of Asset: Minor

The asset contains the severity of a patient's allergies.

Threat Agents:
Hackers looking to know potentially harmful information.

Threats: Possible*Severe

DrugSuggestion.Java

Type of Asset: Data

Class: Other

Value of Asset: Minor

The asset contains the suggested drug to administer to the patient.

Threat Agents:
Hackers looking to change the drug needed:
for personal benefit or physical harm to a patient

Threats: Possible*Moderate

DosingInstructions.Java

Type of Asset: Data

Class: Other

Value of Asset: Minor

The asset contains the instructions for the amount of a drug to give to a patient.

Threat Agents:
Hackers looking to prescribe more medication than needed.
Hackers looking to altar the dosing to dangerous levels

Threats: Possible*Severe

Risks

High Risks

Changing user role and permissions.

Suggested: Give notice to system admin when a change like this occurs.

Low Risks

Hackers trying to disrupt the system or remove patient's information.

What's in place: Backups of the system and information.

High Risks

Name of Threat: Changing medication to a different medication than ordered.

In the code is the function order.java which contains the ability to prescribe or change medication to give to patients. This could be a potential high level threat because if someone got into the system they could change the medicine a nurse would give to the patient to something hazardous to the patient. It could also be used for someone addicted to medication to fake a hospital visit so they could confidently go and have their order changed to some pain relieving medication that they would otherwise not need.

I checked through the code and couldn't find any auditing controls that would keep track of any changes made to the order. What I would change to the program would be to audit it so that if any changes were made to a drug order it would be kept track of when the change was made, from where/what user, and give a notification of some sort of the change next time the patient's information was brought up. This way if any change was made, the nurse giving the medication would be aware that there was a change and could confirm with the doctor of the change before administering it.

I am pretty confident in my findings that there is no audit control for changing the drug order. However, I could not give a 100% guarantee with my findings since I'm not completely sure as to what it could look like so I could have passed it by.

Name of Threat: Hackers changing roles and permissions

In looking through the java files in the openMRS API I found a file called User.java, this file contained the ability to change the roles and permissions of any user in the system. The code also has a isSuperUser function that if activated by simply putting a truth value in then the user has all roles and permissions that come with those roles. There is no auditing that goes along with this and even something as simple as logging when the function was activated would be very beneficial in determining the source of the attack if one were to occur.

Something else I found was that there is auditing in the User file but it is very limited. The only places that it has auditing are the debugging logger which can be shut off and the error logger which only catches the exceptions thrown when trying to convert user roles. It isn't entirely clear to me when the debugging logger is activated but when it does it just seems to take a list of the users roles and not the user or their username. The error logger is along the same lines as the debugger but is a little more thorough in that it retrieves the user and the roles. I am unsure if the apache servers they use to log this have time stamps when the loggers are activated but if not that would also be very useful information to have if an attack occurred.

The User file also contains all of the fields required to create a new user. None of the functions to create a new user seem to have any kind of security measures on or around them so if a hacker were to create an account and somehow activate the isSuperUser function they would have complete control over the roles of all other users on the system. If the program required for an admin to add user then it could greatly reduce the risk of an attack through this area of the code.

I am fairly confident of my evaluation of this portion of OpenMRS but I am not very savvy with the Java programming language so there is a good possibility that I may have miss read or missed a piece of information in the code.

Design Principles

Economy of Mechanism

This principle implies that OpenMRS keeps their web app simple and easy to use for users of all levels. For our group the installation process was anything but simple but in a typical setting where OpenMRS would be used I would imagine an expert would be sent to install it which would in turn make it very simple. Besides that the actual navigation of the website seems to be quite easy to grasp. Whenever a user would want to enter, change or delete any data it is pretty clear where they should go to complete the task. The website itself seems fairly simple to navigate through without too much confusion.

Overall Grade: B

Open Design

Due to the nature of the program OpenMRS will always be open design, which is allowing others to see the code. They suggest others to and are open to any errors or bugs in the program from the public. The program is not private at all and it seems it will stay this way which means the program can not really be anything but purely open design.

Overall Grade: A

Separation of Privilege

Seperation of priveleges makes sure that no one employee or user have all the privleges possible in the system. This makes sure no one takes advantage of any rights they have and prevents one person from having too much power on the system.

E

Least Privilege

The API for OpenMRS did not implement, the design principle, Least Privilege. This means that users of the system had more access to the system than they needed to complete their job. Although this could hurt the system if a user wanted to, it is hard to implement this design principle since there are so many users trying to accomplish different types of tasks.

Overall Grade: C

Psychological Acceptability

The psychological acceptability is a very important principle of the system. If the security of the system was not implemented in a psychologically accepted way than users may try to find ways around the security which compromises the system. Since the system in its current state has no auditing of the API it is accepted as is because there is no inconvenience to the user. However adding auditing would not inconvenience the user any more than without it in place. A simple audit system that tracks any changes made to the system would not inconvenience the user and could potentially even make them feel more comfortable knowing if they made a mistake, their error could be checked on without having to blindly back up the system to how it was.

Overall Grade: A-

Least Astonishment

Least Astonishment is a necessary design principle in any system. A user of the system should never be surprised by an action. OpenMRS does a good job of implementing this design principle into the system. Throughout the API layer the code does exactly what it looks like it should do. If a user does a command to delete a patient, it will delete the patient not the system. OpenMRS is also well documented. If a user doesn't know the action of a command, it won't be hard to look it up. This allows users who don't have a lot of programming knowledge to be able to run commands on the system

Overall Grade: A

Summary of Findings

Searching through the code we found that the code lacked any auditing system in place in the API to keep track of important changes that could be made.This is an important change as every system should have some form of auditing and at the same time would not be too difficult to implement.

Adding auditing to the program would keep it safer and save money in the long run. Without any auditing, if someone wanted to make malicious changes to the program it would make it harder to find what the changes were, and where exactly those changes were coming from. In return this would cost more money and man hours so the time used to implement an auditing system would save more time in the end. It would also prevent innocent mistakes from having as much of a negative affect for users and make the program in general less of a hassle.

Recommendations

Our findings were that implementing an auditing system could control a large number of possible threats, along with a few other minor changes that would not take a lot of work to keep the program safer.

Changing user permissions

The changing of a user's permissions is not something that would happen often. Because of this if a user's permissions were changed our recommendation would be to add a system which would notify a system administrator every time a user's permissions were changed. Along with this should be the name of the user, and when it was changed. If the change was not authorized than the administrator knows that they need to check the system for any changes and make sure everything is secure.

Conclusion

In conclusion, the OpenMRS open source project still needs some work before it will become a globally used system. The API layer has no controls in place for auditing what's happening on in the system. This security flaw needs to be fixed throughout the API layer. Another security flaw that needs to be fixed is user permissions. There should be more levels of accessibility to the system. Users should not have the power they do to harm the system. Although there are still some flaws in the system, OpenMRS is a great project with a lot of possibilities.

API Accountability/Audit May, 2016

Executive Summary

OpenMRS is inadequate when it comes to auditing or keeping track of how users interact with the software. So, it is important that there is an audit system implemented into the program for starters. Now, the great part about an audit system is that you can keep tabs on how the user interacts with the system as stated. Which means, if a user logs into the system it will log who the user is, if the user navigates to the patient's records, the audit system would record what patients records are accessed. Now, after a day or a week or whatever you decide on you can have the log files moved into archive, and when to delete if you wish.

It is also crucial to consider all the threats and threat agents that have been listed. Especially the threats that are listed doomsday, not matter how likely. Because no matter how unlikely it is that a doomsday event will happen, if it does it the end of the company. And even the catastrophic risk should be addressed as a top priority as that would make it so your system would be unusable for at least a month. As a general trend as the risk factor goes down the less urgent the matter is, unless of course the frequency of the occurrence outweighs the fact that it's low risk. Risks are something that should be considered and taken into account with much consideration, as not all can be addressed.

Finally, I would ask you to consider design principles such as separation of privileges while designing the audit system. As you would not want just anyone be able to make changes to the logs. It only takes one bad apple to spoil the rest, meaning, if you let everyone have access to the logs, one person could do nefarious things and change the logs to reflect different. I would also recommend that the audit system would be implemented with complete mediation in mind. Meaning the the system would track everything that is done that way there are no way to do things under the radar. With a properly implemented audit system the security of OpenMRS would go up exponentially with respect to user activity.

SCOPE

We are assessing the API layer of the OpenMRS system and how it audits changes. The API is a Java made API that allows developers to interact with a complex data model using common java objects. The tables in the data model have a one-to-one relationship with a Java object in the API. The Services in the API make sure that only the correct columns are saved/updated in the database. All services are accessed in a static way from the org.openmrs.api.context.Context object. The primary usage of the Context is to call the services so users can fetch and persist data in the database. Authentication and Authorization are done through these services which allow users to make changes in the database; But how does OpenMRS log these changes?

In our assessment we will be taking an in depth look on how does and how should the OpenMRS API audit these changes. For example, if a database administrator goes in the database and looks

up records for a specific person or changes something when they are only suppose to be maintaining the database, the administrators activity should be recorded in detail to avoid security breaches of private information. We are assessing whether the API has the correct ability to log this information just in case a breach does happen, the culprit will be caught.

One interesting piece of OpenMRS is how manipulatable a user object is, especially as a database object. There is a whole piece of code that is devoted to changing and manipulating the user in the database. Where you can change the user's relationship with the database or get the user's relationship with the database. But you can also get the saved properties that are associated with the user, let it be an employee or a patient. You can edit, delete and alter the relationships and roles that the user play. You can also account for the person object itself, and manage how they interact with the program from that respect. It is interesting how the OpenMRS team set up the user objects, to be so malleable and controlled. This helps to account for what people are doing, depending on what relationships/roles they have in the database, will dictate what they can do in the program.

Another interesting feature of OpenMRS is this code
@Authorized(PrivilegeConstants.MANAGE_IMPLEMENTATION_ID)
public void setImplementationId(ImplementationId implementationId) throws APIException;
What this code does is very useful to anyone trying to keep track as to what is going on in the system. It sets an implementation id to a piece of code, and saves it to a database. What can be done with this id is whenever a user tries to do something it can be tracked what they did or tried to do through this id. When a user does/tries to do something they should not do it can then be saved and easily identified.

Another interesting part of the OpenMRS code is the encounters section. The EncounterService.java module basically says, do such and such for said encouter. So, if a user goes into a patient's medical records, this program will have the code that controls the encounter. The code even has the ability to pull the provider of the data, and stores that data. So, now the program gets to tell the person what happens next and grabs who entered the info, this done to keep an audit trail of who does what. It even passes what form was submitted and what type of visit and encounter the provider had with the program. Allowing for a great audit log of what users did and how they did it.

Assets

Type of Asset: Data
Class: SEC
Value of Asset: Minor
Asset:
Probably the most obvious of all assets when it comes down to accountability would be the user’s credentials. After booting up the application, all users are required to enter their credentials. The very first audit on the user is checking if they are who they say they are, using the authentication system. Once the user is in, the program can track their every move, as whatever the user does is now tied to their username.
Threat Agents:

Hacker
Employee
Threats:
Unlikely and Minor:
A key logger would be detrimental to the user that has their credentials logged, as the audit system can not tell the difference between a hacker doing something on the system, and an employee with the same credentials.

Likely Minor:
The user can get themselves into trouble by writing down their credentials, because they can be taken and then used in a malicious manner and will be accounted for as the credential holder not as the person who performs the actions, in which an audit system would not be able to tell the difference.

==========

Type of Asset: Hardware
Class: Other
Value of Asset: Major
Asset:
The database server is essential for an audit system, as the user that logs in is verified they are a user at the database. By having a database, when the audit system grabs say a username and pairs them with an action, if further review is needed they have the ability to pair the the username to a user.
Threat Agents:
Hacker
Power Outage
System Outage

Threats:
Unlikely and Major
A hacker would be a threat agent to a database server, if they are able to perform a SQL injection properly they can change the data in the server, or even delete the data. Make it impossible to trace back a user to a username if they decide they want to change that info.

Unlikely and Major
In the case of hardware, power outages are a threat agent, as if the power goes out and the server is not on a battery backup, it shuts down. Without the database server then there is no way to run the program most importantly, but there are implications to accounting. As there is no way to account for who is logging. And if for some reason the program will let the user authenticate without a database connection, they can do what they want, unaccounted for.

Unlikely and Major
Network Outage would be another threat agent to a database server. The same vulnerabilities for power outages go for network outages. Either the program will not be able to work and auditing will not be an issue. And if the program does let a user log in there will be no way to account for

who is logging in as the username and password are verified. This is not as preventable as a power outage would be, in regards to keeping the server live.

==========

Type of Asset: Data
Class: SEC
Value of Asset: Major
Asset: Patient data is data relating to a single patient, such as his/her diagnosis, name, age, earlier medical history etc
Threat Agents:
A hacker would attack this asset to gain illegal access to data and use it for their own benefits.
A corrupt doctor would attack this asset to use information for their own benefits
Threats:
Unlikely Major: A hacker can gain access to the patient data and delete all the data creating a major slowdown in emergency rooms due to the lack of medical history from the patient.
Rare Catastrophic: A hacker can release private patient data records to the public, creating an epidemic.

==========

Type of Asset: Data
Class: SEC
Value of Asset: Major
Asset: Ip address is a unique string of numbers separated by periods that identifies each computer using the Internet Protocol to communicate over a network. This number can be used to track the location of the person who is accessing the OpenMRS DB.
Threat Agents:
Hackers
Doctors
Workers
Threats:
Unlikely Moderate: A hacker using a proxy like Ip address to access the database and make changes leading to a faulty location in the auditing records.
Rare Not Likely: Doctors/Workers changing their Ip address when accessing OpenMRS, throwing off the correct ip address and location in the auditing records.

Design Principles

There is no auditing system, so instead of assessing the adherence of the accounting system to the design principles, we are just describing how the auditing system should be designed.

Complete Mediation:
Complete Mediation with respect to auditing would be an audit system that keeps a log of everything that happens within the program.

Open Design:
Open design with respect to and API auditing system would be making an audit system that is secure, but not because people don't know how it works.

Separation of Privilege:
Separation of Privilege with respect to API auditing would be who could audit the logs as opposed to those who can't. If there was to be an audit system implemented there would need to be separation of privilege, as you don't want just anyone accessing the log files. But, at the same time you do want the administrators to be able to pull them up if they need to be reviewed.

Least Privilege:
Least Privilege is when you only allow necessary access to asset, so in the case of an auditing system only the employees that need to have access would have it, no one else.

Psychological Acceptability
Psychological acceptability is the conflict between implementing security measures and assuring that the user can still easily access application features. If an application-wide audit system were to be implemented it could be assumed that most features would still be accessed normally. You don't need to announce to the user that logging is taking place so everything will be behind the scenes and completely unobtrusive.

Isolation
Isolation is the task of breaking a system down into subsystems and then securing each of those subsystems. With the implementation of an application-wide audit system we would want to secure each log based on what part of the application is being logged. We can further isolate these logs by applying least privilege. Assigning certain logs to separate administrators ensures that if one account is compromised, not all logs are compromised.

Encapsulation
Data encapsulation is achieved by combining the data and methods needed to interact with it within a single object. Once encapsulated we can implement additional security measures to protect the data. When implemented into an audit system we would hope to see all data encapsulated while being transmitted to the logs.

Least Astonishment
Least astonishment is the practice of ensuring a program works the way it is supposed to. The two perspectives we should be considering in this aspect are that of the application user and the audit administrator. In the case of the application user the audit system should be invisible and unobtrusive. The system shouldn't significantly slow down application use nor interfere with the user's experience. From the administrator's perspective we should see the correct information being directed to the correct log.

Summary Of Findings

In the security assessment of the API risks involving auditing were not handled very well and not at all all together. Comes to find out that OpenMrs doesn't have any auditing system to prevent

these risks or to log changes. We find that it is easy to get away with making changes to the data inside of the database without anyone else knowing who made the changes and it is very easy to delete data too.

Recommendations

The OpenMRS community should focus on implementing an audit system. If there was to be an audit system implemented there would need to be separation of privileges, as you don't want just anyone accessing the log files. But, at the same time you do want the administrators to be able to pull them up if they need to be reviewed. An Audit system should have Least Privileges as well to allow necessary access to asset, so in the case of an auditing system only the employees that need to have access would have it, no one else. Only the people who need access to the audit system will have access. All the other people will not be able to make any changes that are related to the audit system. When implementing an audit system it's important to prioritize what the audit system is logging. Considering this software is typically used in areas of the world without access to a ton of resources, it may not be possible to reliably log every interaction with the software. When developing an audit system it would be ideal to look at all interactions a user could have with the system and then categorize them into different priority levels. Those levels of priority could then be coupled with system requirements to form an application that matches the level of auditing with the resources at a particular location.

It's important to remember that an audit system isn't just about logging data but also raising flags when suspicious activity is detected:
Rapid password attempts
New access to root
Attempts at accessing files that the user has no reason to
Attempts at executing codes in input fields
These are just a few of the actions that should raise some sort of alarm within the system.

Conclusion:

The lack of an application-wide audit system is definitely a huge concern. As stated before we were able to easily change data in the OpenMRS system. Without an audit-log system in place, anyone with malicious intent could change patient data. To one extent it's possible that this could aid in theft as anyone could "prescribe" a patient any amount of a drug and there would be no way to prove who did it. It's often difficult to plan to this extent, however, implementing a system to catch the basics will in turn help prevent situations like this.

API Confidentiality May, 2016

Executive Summary

OpenMRS is a great program that is very beneficial to medical facilities in remote areas all around the world. Being an open source software, their take on security isn't the best. They are currently working on implementing into their systems a greater focus on roles and permissions to ensure the privacy and confidentiality of information. Roles and permissions restrict users of a system so that they cannot view patient information that they don't need access to.

OpenMRS's privacy policy is to "Please assume that everything you contribute intentionally to OpenMRS is public." They assume that users have a mutual respect and because of this allow a lot of information to be public. "This includes emails on public lists, wikis, online discussions of all sorts, and software or documentation that you post to this website." These items are primarily protected by patents and copyright law. Having all of these items being available to the public could create security concerns and could lead to the accidental release of private information.

The API layer has many different Objects that contain information on patients as well as users of the system. These Objects need to be secured to ensure the confidentiality of personal information. If these Objects become compromised, information such as Names, Addresses, Medical Records, Social Security Numbers, and much more could be stolen. The threat for this is reduced in remote areas where OpenMRS is prevalent, but with the software growing to new parts of the world it faces new threats.

Scope

The Privacy Policy of OpenMRS is to "Please assume that everything you contribute intentionally to OpenMRS is public". Anything posted on the OpenMRS webpage such as emails on public lists, wikis, online discussions, and documentation are all public information. If you want some of this information to stay confidential, you can choose not to share with OpenMRS. OpenMRS can ask you to share private information but without going through a legal process, they can't get access to your information if you don't want to give it to them.

The API layer of OpenMRS gives the developers a way to interact with the OpenMRS Data Model. This is done through the use of Java. In the API layer, there are many different objects to hold patient information and other medical records. Some of these objects include the Person Object, the Patient Object, and the User Object. Each object contains different information about a patient or a user of the system.

The Person Object can be used to get data on a person. A person object in OpenMRS can be a patient or a user in the database. The person object contains data such as, birthdate, gender, age, ID, name, address, etc. A single person can have multiple names or addresses. Additional data is added to a person based on where the install location is. You can for example search for an ID number using the person object. This is accomplished by using the PersonService. Example Code is below:

```
PersonService personService = Context.getPersonService():
Person p = personService.getPerson(1):
```

Also with the Person Object, you can update data on a person. This is also done by the PersonService. Example Code is below:

```
PersonService personService = Context.getPersonService():
Person p = personService.getPerson(1):
p.setGender("M"):
personService.savePerson(p):
```

The Person Object holds most of the patient data and is linked to the Patient Object. The Patient Object inherits attributes and objects from the person object it is connected to. The patient object has an ID and PatitentIdentifiers as well as the attributes and objects it inherits from the person object. PatientIdentifiers can be used to find a patient in the same way that PersonService is used to find a person. The identifiers are set by administrators and can include different properties to be identified such as a location or a number.

The trouble with Confidentiality in the API layer is that they have to make sure each object in the API is secure and that people who shouldn't be able to access certain information can't access it. This is done primarily by assigning roles and setting permissions. Roles have assigned permissions to them and ensure that a user can't see medical records and patient information that they don't need access to.

The User object is made for people who can access and log into the system through the use of a username and password. These users are given permissions to restrict what they are able to access. It is important to be strict with these permissions to ensure that the data is kept confidential and privacy laws are upheld where the system is deployed. Encryption can also be used to ensure that confidential information isn't in plain-text to be easily viewed.

Medical information is protected through agreements with vendors and their business partners. HIPAA as well as state and federal laws are used to enforce confidentiality of medical information. According to OpenMRS, confidentiality of a patient's data should be ensured and controlled by the patient. The ideal situation for confidentiality would be that only the patient and whoever they authorize can view their medical information.

Assets

#1 Patient Identifier

Type of Asset: **Data**

Class: **PII**

Value of Asset: **Moderate**

Patient identifier is used to query patient record for a particular patient through the OpenMRS API (See an example code in Scope section above).

Threat Agents:
Malicious users or organizations who are interested in a particular patient's medical records or medical treatments or personal information such as birthdate, blood type and patient's location.

Threats:
It is very likely that a malicious user will be able to get a particular patient's identifier and use it to attempt querying for the patient's record using the OpenMRS API. It is unlikely that the malicious user can get access to all of the patient's record since the OpenMRS has implemented user authentication and authorization mechanism to set permission level to each user what he or she can do through the OpenMRS API. For this reason, the value of patient Identifier is Moderate.

#2 Patient's Medical records and personal infomation

Type of Asset: **Data**

Class: **PHI**

Value of Asset: **Critical**

All of the medical records of patients in OpenMRS database.

Threat Agents:
Malicious users or organizations who are interested in patient's' record in the OpenMRS database.

Threats:
It is unlikely that malicious users can get access to patient's record in the OpenMRS database through the API layer since the API layer implements authentication and authorization to access patient's records. The malicious users would have to come up with a way to get around that security policy. Should the malicious user gets access to patient's records in the OpenMRS database, severity of the attack is critical and the impact is detrimental to the existence of the OpenMRS software.

#3 Username & Password combo to authenticate users

Type of Asset: **Data**

Class: **SEC**

Value of Asset: **Critical**

Username and Password to authenticate a user into the system, authenticated users are given permissions to restrict what they are able to access.

Threat Agents:
Malicious users or organizations who are interested in patient's' record in the OpenMRS database.

Threats:
It is very unlikely that a malicious user gets access to both other user's username and password since there's a permission level set in the Database layer and passwords are hashed before persisted in the database. However, if the malicious user managed to get both username and password for a user who can change the restriction level for what the user is able to access, the severity of the attack is critical.

#4 Order

Type of Asset: **Data**

Class: **PHI**

Value of Asset: **Critical**

The order that the users save into the database. This order contain high value information about patient's condition and drug

Threat Agents: Person who attempts to harm to other people's health

Threat: If there is a change made to the order in the database, the patient likely receives a different medication and may result lethal damage or even death

---

#5 Location

Type of Asset: **Data**

Class: **PII**

Value of Asset: **Moderate**

The location of the user is saved into the database system

Threat Agents: Drug corporations that want to regulate the price of drugs

Threats: If the drug corporations know where there are a lot of patients using OpenMRS in an area, they're likely gonna increase the price of drugs in that area. Countries which use OpenMRS are developing, they want to approach the new technology, drugs for better life so they are a tasty targets for these corporations.

#6 Visit

Type of Asset: **Data**

Class: **PII**

Value of Asset: **Minor**

The visit time/location is saved into the database. Usually, the doctor allow anyone can visit their patients in a frame time, otherwise the patients are kept resting quietly

Threat Agents: malicious people who want to harm other people by disrupting the visit time frame

Threats: Once the visit time frame is disrupted and fixed, the patient's healing process will be delayed or the information of patient's family will be leaked.

#7 Medical Insurance

Type of Asset: **Data**

Class: **PHI**

Value of Asset: **Moderate**

The medical insurance provider information and coverage is an asset that is valuable to the facility. Medical insurance information provides information about coverage, copays, and other important information needed by a treatment facility. This is also important for billing the insurance and keeping this information confidential is important to uphold privacy laws.

Threat Agents:

A possible threat agent could be someone trying to pass off as someone else.

Threats:

If they obtain your medical insurance information and enough information about you, they can possibly use your insurance. This could be useful if someone doesn't have medical insurance and can gain access to these records.

#8 Prescribed Medications

Type of Asset: **Data**

Class: **PHI**

Value of Asset: **Moderate**

Prescribed medications can be helpful to see what a patient is currently taking to make sure that you don't prescribe them a medication that will react badly with what they are taking. Also, it can be helpful to see what they have been on and what works for them.

Threat Agents:

A threat agent could be someone looking for a way to get medicine or prescription drugs.

Threats:

They could attempt to find out who is prescribed controlled substances or medication that they want.

They could use this information to somehow get their hands on a drug that they shouldn't have access to
Medical Records, Social Security Numbers, and much more could be stolen. The threat for this is reduced in remote areas where OpenMRS is prevalent, but with the software growing to new parts of the world it faces new threats.

#9 Database

Type of Asset: **Data**

Class: **PHI, PII, SEC**

Value of Asset: **Critical**

The database that has all the information of OpenMRS

Threats Agents: Malicious SQL attackers

Threats: Attackers can launch an SQL injection from API layer directly into the database system and manipulate a whole system.

Risks

Malicious users toward patient identifier asset:

- Controller: https://github.com/openmrs/openmrs-core/blob/master/api/src/main/java/org/openmrs/api/PatientService.javaThe controller reduce the probability of the threat being attempted, if the Id and password are null or dont match, it returns error
  - Missing control : there is no control that delete the id and password after the user log out, it may lead another person get access into user's information
  - We are not very confident in this this assessment. We dig around this controller and found no delete command to the Id and password. We may not know much about java but base on what we know about ID and password algorithm , this controller is not complete.
  - Impression: OpenMRS did really well on controlling the risk to this asset. Obviously, this is a basic barrier that they have to put up to prevent a very basic attack to the system. Developers sometime focus too much into important attacks but forget some easy mistake from the entrance can lead to serious consequences.

Attack to personal information:

- Controller: https://github.com/openmrs/openmrs-core/blob/master/api/src/main/java/org/openmrs/api/PersonService.javaThis controller reduce the personal information of patient being accessed, restrict the access. Depend on relationship to the patient, it will decide who have access.
  - Missing control: None
  - We not very confident about this assessment. We tried to see if is there any kind of attempt can be made to harm this asset, but overall we didn't see anything yet. The personal information is restricted to the doctors and patients only.
  - Impression: OpenMRS did an amazing job here to control the access to patient's personal information. It is essential and important step to take to secure the confidentiality. Also, the controller has command to remove any user who tried to break through and steal the information

Password Threats:

- Controller: https://github.com/openmrs/openmrs-core/blob/master/api/src/main/java/org/openmrs/api/UserService.javaThis controller reduces the probability of an attack being successful. If a person want to change the password of a user, he/she need to know the old password. Any attempt trying to

delete the user's information in the database will be violating foreign key constraints and fail.

- Missing control: Controller on people with privileges, they usually get exception command from every controllers; through this, any top admins can manage the password which violate the confidentility
- We are not confident about this assessment. But the exceptions in the controllers make us suspicious about the power of admins. What happen if they create a backdoor on the password system.
- Impression: It's great to see the petition to users who attempt to violate the password system. It make us feel more secures about how our password is protected very strictly.

Mistake to Order

- Controller: https://github.com/openmrs/openmrs-core/blob/master/api/src/main/java/org/openmrs/api/OrderService.javaThis controller minimum the threats to the Order that was made and saved into the database. Mistakes that are made on the order will not allow the order to be processed through the system. Some function relate to order like revising the stopped/expired order.
  - Missing control: None
  - We are confident about this assessment. This is a minimum threat so we dont need to peek around too much. The controller is there just like a reminder to person who fill the order that some things should be done correctly
  - Impression: It's a good controller to remind people about their mistakes that that make on the order. And mistake on person's life is not acceptable. OpenMRS will not validate any mistake. It also prove many other functions to keep the order easier to interact with.

Mistake to Location

- Controller: https://github.com/openmrs/openmrs-core/blob/master/api/src/main/java/org/openmrs/api/LocationService.javaThis controller throw APIException if the location para is null
  - Missing control: None
  - We are not really confident about this assessment. The locationservice looks vague due to our skill in java is not good enough to actually understand what it does to control serious threat. But look like it make a really good reminder that null parameter should be filled.
  - Impression: Again, OpenMRS did well on controlling minimal threats to the system. Here we are talking about UI parameter of location. It's better if they add some comment so that people could understand the controller easier.

Milking medical insurance

- Controller: https://github.com/openmrs/openmrs-core/blob/master/api/src/main/java/org/openmrs/api/ProviderService.java

This controller grant access to provider with the correct ID the information about the medical insurance. It throws APIException if the identifier is null, duplicate and blank string.

- Missing control: None
    - We are not confident about this assessment. We dig around the controller a few times but hardly understand it. We are not sure what we are saying about this controller is right or not.
        - Impression: The Exception about duplicate is a really good one. It prevent the attackers create a duplicate account to take advantage of the insurance. It make the threat agent reveal himself if he tried to be the owner of the insurance.
        -

Design Principles

Economy of Mechanism:
If the design and implementation of security is simple, there will be a less likelihood of error to occur.
Most of the security I found in most of the services had similar ways of approaching different tasks. Simple design was noticed, as I saw how information that is entered blank has apiexceptions to handle the errors. Also if information is entered twice, the original tag gets replaced by the duplicate. There might be more ways around OpenMRS security, but the design helps create less confusion when the problem must be addressed from what I can see.
Grade A

Open Design:
Open design is the development of physical products, machines and systems through use of publicly shared design information. Openmrs is at its strongest when the public is allowed access and can create code that can be edited and fixed by others. Of course because all of its contents can be seen by the public, the use for Openmrs is limited but effective in certain environments.
Grade B

Isolation:
Determines how transaction integrity is visible to other users and system
Even though Openmrs is an open source, you still need authority to access the content that is being kept away from users. Even though I feel that the user who has authority has too much access to different objects, the public and ones privileged are separated properly.
Grade C

Layering:
Layering is the organization of programming into separate functional components that interact in some sequential and hierarchical way, with each layer usually having an interface only to the layer above it and the layer below it. All of the code use a top down layout and separate from each other when different task are needed. It was readable and will organized, which helps promote community contribution and understanding.
Grade B

Recommendations

Malicious users toward patient identifierThis threat is well controlled by the OpenMRS but it still has a breach. It doesn't have the command to delete the Id and password after the user log out the browser. It's dangerous if someone take advantage of this breach. So, the solution for this, I think we should add a command to secure the Id and password after logging out. It's likely the user has to type the Id and password every time they log into the account, but it becomes more secure.

Attacks to personal informationThis threat violate the privacy of patient's information. If someone pretend to be a doctor of the patient, as they create a fake relationship with the patient, they have access to the patient's condition and drug he/she uses. To solve this problem, we need to be strict on the person who has relationship with the patient. Only the doctor who is responsible for the patient has the right to give the access to another person and every actions need to be saved into history for further investigation if anything wrong happens.

Password ThreatsUserservice controller gives admins privilege/exception to change a person's password or look at the password. If that admin tells his friend to exploit that information, it would be really harmful. The solution for this problem is limiting the admin's power to regulate the user's password. If the user forget the password, the admin can send a code for the user to reset the password without looking at the password. We can also apply the separation of privilege so that no admins can take advantages of his power.

Mistake to order Overall, there is not much threat to the order form. It's better if OpenMRS add comment every time the user click on the form, letting him/her know that how this form should be filled. That's a better approach to UI interface. The order once they goes through the API layer, they should be checked if they has any sign of suspicious code that can interact with the source code exception, because that's likely a hacker who discover a breach inside the source code.

Mistake to locationA threat that relate to location, it's likely a leak of user's location. If bad guy know that the user is not home, he may pay a visit. So solve this threat, we could give the user options whether he/she agrees to share the current location or not. If not, there will be a command that request interface leave the location parameter blank.

Milking medical insuranceWhen exploiter has the same access to the insurance information of the patient, he can manipulate and take advantage of that right. Hence, we should solve this problem by alerting the patient that their insurance's right is being violated by the third party. It's like when we spend money with credit card, an email will be sent to us. Further step

Mistake in Prescribed MedicationsPrescribed medications can be helpful to see what a patient is currently taking to make sure that you don't prescribe them a medication that will react badly with what they are taking. If someone make a change on prescribed medications, it will hurt the patient as he/she has to take the medication that he/she are not supposed to take. The solution for this problem is implement a history of change in the prescribed medication so the doctor will know what changes have been made in the system, why they happen, when and where. If the

change is significant, we need to tackle down the Id of the malicious user who made changes to prescribed medication.

Conclusion

After analyzing API Confidentiality layer of the OpenMRS software, we came to a conclusion that the OpenMRS API has some security concerns around its implementations of some of their security policy. Complete mediation is needed for some of the assets we listed in section 2. However we found that the OpenMRS is lacking in some of the security area around those assets.

Overall we found that the OpenMRS provides values to people in the medical fields, especially people who would not be able to have access to these valuable data and records otherwise. However we think that the developer community around this open source software needs to go over some of the security concerns of the software.

OpenMRS Database: Authorization May 2015

Scope

Our group will be assessing the OpenMRS database. The database is where a system's data is stored. This includes information such as patient records, names, addresses, and other medical information. MySQL is the database system that is utilized. Considering that this is the area in which medical information is stored, security is of extreme importance for this part of OpenMRS.

Specifically, we will be assessing the authorization process of the OpenMRS database. Our goal is to determine whether current authorization practices or techniques in use are satisfactory. We will use HIPAA guidelines as a reference point to determine how effective the authorization process for this database is. We will examine what the process is for being authorized to access the database. We will also determine how safe the database is from certain attacks and how the authorization related processes can be improved and optimized. It is important to evaluate any authorization processes that are currently being utilized.

Overall, we will be examining ways in which the authorization process for the database can be improved. We're also looking at where it is vulnerable. If there is a single entry point to the database or flaw in the authorization process the results could be catastrophic. Considering that there is medical information in this database it will certainly be a main target for attackers. Examining these processes with the HIPAA guidelines in mind could be very beneficial to improving the database security.

OpenMRS Install Instructions on Windows

1. Check System Requirements
2. Install Firefox
3. Install Java, They recommend Java 7, but this did not work for me I had to download Java 6 (Java 8 is not supported)
4. Install Tomcat (see section below "Installing Tomcat for OpenMRS on Windows)
5. install MySQL (see section below "insatlling MySQL for OpenMRS on Windows)
6. Deploy OpenMRS (see section below "Deploying OpenMRS on Windows)
7. Configure OpenMRS (this section is not in the OpenMRS Wiki and skipping this section the application works fine but may not be correct for real world use)
8. You have finished your installation

Installing Tomcat for OpenMRS on Windows

1. Make sure you have Java installed beforehand.
2. Download Tomcat http://archive.apache.org/dist/tomcat/tomcat-6/v6.0.29/bin/apache-tomcat-6.0.29.exe
3. Start the Install
4. Accept the Destination Folder
5. Accept HTTP/1.1 Connector Port 8080

6. Accept Java Directory Detected (Make sure it is Java 6 or Java 7)
7. Select Install Tomcat
8. Go to C:\Program Files\Apache Software Foundation\Tomcat 6.0\conf
9. Locate the file "tomcat-users.xml" and try to open it.
10. Open as administrator tomcat-users.xml in a text editor (I recommend Notepad++ or any text editor that shows you line count)
11. Edit the 18th line with the following code

1. Once this is done save it and you are finished with the tomcat installation

Installing MySQL for OpenMRS on Windows

1. Download latest MySQL https://dev.mysql.com/downloads/installer/
2. Run the Install Program
3. Accept the License Agreement
4. If asked to update installer make sure you accept it
5. Choose Full Installation Setup and make sure you choose the correct architecture (x86 is 32 bit x64 is 64 bit)
6. You will be shown what needs to be on your machine in order for MySQL to work, make sure everything is installed correctly before continuing
7. Choose the Developer Machine config choice
8. Leave the rest Default
9. Create the Root Username and Password
10. Finish

Deploying OpenMRS on Windows

1. Make sure Tomcat is running (It will be on your tray as green)
2. Download the latest stable version of OpenMRS
3. Navigate to http://localhost:8080/manager/html and enter the Tomcat Root credentials
4. in the Tomcat Web Application Manager enter the location of openmrs.war file to deploy (this should be found in your recent downloads)
5. Once it is done deploying refresh the page you are on and /opensmrs should be displayed under Applications. (also Tomcat should start running the application as well)

OpenMRS Install Instructions on Mac OSX

1. Check System Requirements
2. Install Firefox
3. Install Java & Tomcat
4. Install MySQL
5. Install OpenMRS

If you don't have the prerequisites installed on your Mac

1. Download and install Firefox here: https://www.mozilla.org/en-US/firefox/new/.

2. Install Java & Tomcat by following these instructions: http://wolfpaulus.com/jounal/mac/tomcat8/
3. Install MySQL by downloading the DMG from this link(choose Mac OS X as platform): http://dev.mysql.com/downloads/mysql/
4. After you have installed all of these, refer to my instructions below to get OpenMRS up and running.

If you do have all the prerequisites installed (My install, I had everything installed from prior classes)

In your web browser go to http://openmrs.org. In the top right click the Download button. On the top of this page it will tell you the most recent and stable version. I chose openMRS 2.2 standalone edition. After downloading, open the folder that you have downloaded. Then I opened the README which told me how to install it for Mac OS X. All I had to do is open the "openers-standalone.jar" file and it gave me two options for the install. I chose the option to use "demonstration mode" because we are new to openMRS and this already gave us a database to work with.

A page will pop up in your web browser after the OpenMRS server is up and running and the default username is "admin" and the default password is "Admin123". Then choose which location you are at, for ours we picked the "Inpatient Ward"

Assets

Login Credentials

Type of Asset: Data

Class: SEC

Value of Asset: Major

This is their username and password. The OpenMRS install uses the default username "admin" and default password "Admin123".

Threat Agents:

- Employees
- Other companies
- Anybody "shoulder surfing".

Threats:

- **Likely*Major** - A hacker trying to login to systems with the credentials "admin" and "Admin123" because that is the default. They are bound to get in to a system with OpenMRS if they keep trying.

- **Likely*Major** - A hacker once logged in with the default name and password is granted super user privilege. They can then delete authorized users from the database and this could deny access to them.
- **Possible*Moderate** - Once logged in a hacker can go to the advanced settings, and change the roles and privileges of other users. They can give more access to files and features that some users shouldn't have, or they can decrease and restrict their access.
- **Likely*Major** - Once logged in, the user can view patient records and violate HIPAA. When OpenMRS is installed, the super user username and password should be changed t prevent this. This is a pretty serious security flaw if an inexperienced person is installing OpenMRS.

Database Servers

Type of Asset: Hardware

Class: Other

Value of Asset: Major

Any company servers in which data or database data is stored.

Threat Agents:

- Employees (Insider attackers)
- Outside attackers

Threats:

- **Possible*Major** An employee accesses the database servers from within the company and copies the data onto a storage device, stealing the data.
- **Possible*Major** A disgruntled employee accesses the database servers from within and deletes, changes, or manipulates data in some way.
- **Possible*Major** An outside attacker utilizes the admin and admin123 username and password weakness to gain access to the database servers. The attacker manipulates, deletes, or steals the data.

Company Routers

Type of Asset: Hardware

Class: Other

Value of Asset: Major

The routers used by a company that uses OpenMRS. These would presumably be used by employees. They potentially allow access to the database, API, or Webapp.

Threat Agents:

- Employee (inside attacker).
- Someone who gains entry into the company's place of business.
- Outside attacker.

Threats:

- **Possible*Moderate** An employee finds the password to the router underneath(if he/she didn't already know). That employee then uses that information to try and access parts of the system he/she shouldn't (with or without success) or uses a router traffic sniffing program.
- **Possible*Moderate** A person wants to learn the password to the router within a certain company. He or she finds a way to enter the building or gets a maintenance worker(someone within the company) to look at the bottom of a router being used by that company. With this information a person could access the router outside of a building and use a traffic sniffer to read the traffic on the router. This could potentially lead to a greater security breach.

Java Requirement for Installation

Type of Asset: Software

Class: Other

Value of Asset: Moderate

During the installation process and initial running of OpenMRS, an error is received if you do not have the right Java program installed on your computer. In order to run OpenMRS you need an older version of Java.

Threat Agents:

- Attackers.
- Breakdown of program.
- Outdated software

Threats:

- **Possible*Moderate** If any attacker is looking for a weakness in the security of a system, they could look at the Java version required for OpenMRS. Given that it is an older version it's security may not be as updated or effective as a newer version. Also, since it is an older version, the chances of it having known vulnerabilities is also higher.
- **Unlikely*Minor** The system could have errors occur due to the age of the Java version they are using. If other parts are more up to date versions of software, then eventually you may be able to use the older versions of software with the newer versions you are using in other parts.

Failure or Destruction of Database

Type of Asset: Hardware

Class: Other

Value of Asset: Major

The database servers hold valuable information. Their components are also prone to breakdown. Data is usually saved onto some kind of disk or tape within the database hardware. These disks or tape can malfunction or just break down. A natural disaster (fire, lightening storm) could also destroy the storage hardware.

Threat Agents:

- Break down or malfunction of database hardware. (Disks or tape, ect)
- Natural disasters(fires or storms)

Threats:

- **Likely*Minor** One of your storage disks or other database hardware could malfunction. Hopefully there is a back-up disk in place for such an occurrence. This isn't an unlikely occurrence but if there are proper back-ups in place it shouldn't cause much of an issue.
- **Possible*Moderate** A fire or some other type of natural disaster could occur at the site of the database hardware. This could destroy your database, and other assets within your company. If proper back-ups of data are maintained then this isn't as severe of an issue. Back-ups can be costly to maintain.

Patient Registration Information

Type of Asset: Data

Class: PII

Value of Asset: Major

Any of the information that may have been entered with the registration of a new patient such as the DOB, telephone number, name, and address.

Threat Agents:

- Users of the system
- Outside attacker
- System Failures

Threats:

- **Possible*Minor** Someone sells the registration information.
- **Possible*Moderate** Someone uses the information to do harm to a patient.
- **Likely*Moderate** A Hardware malfunction could cause loss of the data needed to contact a patient.

- **Likely*Minor** Someone could modify the information incorrectly.
- **Unlikely*Moderate** Someone could delete the information unintentionally or otherwise.

Permissions

Type of Asset: Data

Class: SEC

Value of Asset: Major

The type of access that each individual user is allowed to explore within openMRS. An admin will have more permissions than an employee, and an employee will have more permissions than a regular user.

Threat Agents:

- Someone changes the permissions of another person.
- Permissions are given to a certain personnel that were not meant to be in place.
- Permissions were changed to have different types.
- Permissions were changed but not correctly distributed,

Threats:

- **Likely*Minor** A user is given permissions that they were not supposed to have and they have access to something that they shouldn't be able to view in the first place.
- **Likely*Moderate** An attacker has compromised the permissions that are placed on the database for ill-intentions and is using the permissions to access areas or lock employees out of certain areas within the database.
- **Possible*Major** An attacker has found a way to give themselves the correct permissions in order to take information and stockpile medical records, Identities and other personal information in large amounts over a period of time.

Risks

Extreme Risks

Default Credentials on Administrator Account when first setting up the Database.

While Assessing the OpenMRS Documentation I saw no mention to the Default Username and Password and how to prevent this security flaw from being used. If an inexperienced user were to install OpenMRS and use it in their Clinic without changing the credentials the possible results could be detrimental. Though with that being said it is basically self-explanatory for the user to understand why they should change the credentials in the first place.

The controls that are missing from the OpenMRS Documentation are all of them. The Documentation does not state within it that the users should change the credentials after their

first log in. There is not controls on how to *prevent it*, how to *stop it from being successful*, or even how to *reduce the damage it would do.* With the default account being basically the equivalent to the root account on a Linux machine it would have full access to all the information stored within the database.

With my assessment I was very thorough with my research through the documentation, I even used the search function they have built into the documentation to see if anyone had even asked about this topic, no one has. With this risk I was very understandable of what exactly I was looking for but to no avail I was unable to find any sort of documentation on the matter.

This threat is an easily fixed one that OpenMRS has no excuse not to fix in the first place. The OpenMRS team is being plain lazy when it comes to this threat and the potential repercussions of this risk are in the *Extreme* Range. If the users forget to change the default administration Username and Password after setting up the result would be catastrophic, not only resulting in Personal Identifiable Information (PII) to be released, but also Personal Health Information (PHI) to be taken and possibly used with malicious intent. The fix for this problem is easy at best, all they have to do is implement a forced instance that makes the user after logging in for the first time create new credentials to become the default admin account, and delete the old standard one. With this fix the users would have no way of mistakenly forgetting to change the credential eliminating the risk all together.

High Risks

Using an outdated version of Java to run OpenMRS.

This threat does really need a specific control. OpenMRS would need to be able to support the newest Java version in order to correct this issue. It may be a difficult change, but it's necessary if one wishes to maintain security. Upon trying to download the older version of Java you are shown a warning about the vulnerabilities associated with using an older version of Java. It lacks newer improvements on algorithms and measures. Since the older version of Java has been available longer, attackers have had time to find vulnerabilities in the system. Overall, it would probably be easier to exploit.

I'm confident in the assessment because of the information on the Java webpage. A clear warning is given that using an outdated version of Java is a security vulnerability. It is not recommended. Not to mention the potential compatibility issues with newer software used within the system. It's also safer practice to stay up to date on all parts of your system hardware and software included. Newer software and hardware is improved and updated in terms of performance and security. Whether it's a simple bug fix or an implementation of a new security system.

Design Principles

Economy of Mechanism

This principle is to stress the point to keep the authentication and authorization system simple. This principle applies to all aspects of the software but more importantly to the protection

mechanisms. When designing the authentication system, unwanted access may go unnoticed because it is being tested with normal use. A system must be put in place to recognize and check the input at login, and this must be done with a small and simple design for the most successful outcome.

C It has room for improvement due to the fact that we could not see what the authentication code was doing in the background. To authenticate, context is used. An example is "Context.authenticate('Bob','password')". This is very simple and good for economy of mechanism, but we can not see how Context is actually doing the authentication. We found a complaint about how slow this method is which may mean that it is not very simple and efficient. Here is the page with the complaint: https://wiki.openmrs.org/questions/77332739

Fail-safe Defaults

This principle is that software should come with already set defaults that should not lead the user to a failure. These are just base decisions on permissions, and this is very important to implement correctly. When is comes to the authentication system, you would want to set default security measures, one of which may be to require the user to set a admin user name and password and not give them a default one.

C This needs improvement due to the fact of the default username "admin" and the default password "Admin123". This is NOT a fail safe default. The failure this could lead to is the fact that attackers would use this username and password as their first attempt, and will probably succeed since some installs of OpenMRS may have never changed these defaults.

Complete Mediation

Complete Mediation is making sure that every single access to any object in the system must be checked for authority. A single unchecked access could be the breach point for an attack. This means that each connection to the database should have an authorization check and each action should be checked for authorization. It only takes one failure to check and the system becomes vulnerable.

A It is stated that before every method call, the currently-authenticated user's privilege is checked. This seems to be implemented in the API layer. As for the database, it is said that basic SQL CRUD permissions are on each table which implies that they are checked per access.

Separation of Privilege

Separation of privilege is the idea of separating users privileges and roles so that not one user alone can compromise protected information. Its the idea of having multiple keys rather than one and this is important wit access to the database. Nurses should have different privileges than doctors, and if they want to access information that they shouldn't, they would need people with other roles to help them breach this.

A We give them an A because of how many roles and privileges they have automatically defined, and the fact that you can create new ones as well. No one user can access information they shouldn't if their role is configured properly.

Least Privilege

Least privilege is very important when it comes to the database. This is the idea that every user of the system should operate using the least set of privileges necessary to complete their job. When dealing with all of he users that login this system, this is very important because this roles vary from nurses and doctors, to data clerks. The roles assigned are meant to give the least privilege.

A I gave this an A for least privilege because it has the option to set roles and privileges based on the users position. There is a system for inheriting roles so you don't have to completely redefine the roles for each user. You can also set special privileges for certain people and take some away from others if their default does not satisfy your needs. There is also the option to add your own defined roles as well if the 27 predefined ones do not match your needs.

Psychological Acceptability

This means that the security measures that are in place do not interfere with an authorized user using the database system. The system security should maintain that confidentiality and integrity of the database. It should not interfere or hinder the work of authorized users accessing the database. An example of this would be an authorized hospital employee trying to access the database. He or she should be able to input their log-in credentials and access the database with no issues. At the same time the database security must be maintained.

B The system in place seems to be acceptable. It is simple enough to input log-in credentials and gain access to the system and the database. We did not notice anything that would hinder our work within the database. It does not take a significant amount of time to access the system and database. A user would not be frustrated by using this system. A user would also not find that the system interferes with their work unduly. Some small adjustments to make the system easier to navigate could be helpful.

Isolation

The idea is to split a computer system into smaller pieces and make sure that each piece is separated from the other ones, so that if it gets compromised/malfunctions, then it cannot affect the other entities in the system. However, techniques that have been very useful in improving the security of individual computer system does not work very well in the network environment. Another problem is how to partition the system into meaningful pieces and how to set permissions for each piece.

C-There wasn’t a lot of information that was available if isolation does apply to openMRS. We don’t believe that they are implementing isolation into their security due to it’s complicity. However there doesn't seem to be a need to implement isolation.

Modularity

To be modular is to be broken up into smaller, easily update-able pieces. This is absolutely essential in a large scale project and even more so if the project is open source. Modularity allows for much more manageability by separating the program into several smaller ones that can each be used in separate programs. It generally also makes maintenance less of a chore.

B OpenMRS utilizes many different modules effectively. The main issue is that, at least for the 2.2 standalone, the latest run-time environment of java is not supported. This suggests that at least some of the modules need to be updated and that there may be an issue regarding the maintenance or dependencies of the problem modules.

Least Astonishment

This principal means that whenever a user uses the system, they should not be confused or surprised by the response of the system. In this context, the user should be able to use the system and database with few surprises. When the user gives input, he or she should not be surprised by the action that results. When a user adds input to the database or accesses the database, it should respond as the user would expect. This isn't easy to obtain, but it makes your product more usable by users.

B We found the system responds the way you would expect. The database also acts and works as you would expect it to. They function well in this regard. A user would not be surprised by the responses of the system when providing input. Operating within the system and database is straight forward, with little surprise as to what each control does. Some work could be done to make the install of OpenMRS easier. This could use significant improvement.

Summary of Findings

The design principle of least astonishment was violated by OpenMRS. I would point to the installation process as a big part of this violation. In some cases installing OpenMRS went fairly smoothly without any surprises or errors. However, depending on what operating system you were using you may have some unexpected surprises and trouble with installation. I know that if you are installing on Windows you have to make sure you have the right version of Java. This also happens to be an older version of Java. For someone with experience working with computers and installing software this might not be an issue. You have to consider novice users though. Any extra steps outside of normal installation would be a surprise and probably a hassle for a novice user. This is exactly why least astonishment is an important principle. If users are surprised by extra install steps, they may become confused or disheartened. If that's the case then they probably won't use your software.

The design principle of fail safe defaults was also violated by OpenMRS. Although not a threat to usablilty, the use of a default superuser login and password is a large issue concerning security. It is best practice not to have a Admin login and password that is the same across all default installations. It needs to be considered that someone, if given the option will leave the

defaults as is which makes a security breach as simple as someone trying the defaults on the systems the come across.

The design principle of economy of mechanism was violated by OpenMRS. It has room for improvement due to the fact that we couldn't see what was the authentication was doing in the background of the database. There also have been complaints about how slow this method is which may mean that it isn't simple or efficient to have implemented. So this aspect of authentication and authorization may not be working as intended and may need improvement in order to meet the correct specifications.

Recommendations

Limit failed login attempts

Brute forcing with a dictionary is a common method of attack. One way to limit the success of such attacks is to limit the number of times an ip can fail at logging in. Possibly even an account should be monitored instead of ips. If there are many failed attempts at logging in as the same user there should at least be an automatic warning if locking that users account is out of the question.

Make a unique password upon setup

Using a default login/password is terrible practice for security because it's likely that some people will leave it as is despite being told they should change it. Instead of using a default login and password, whoever is setting up OpenMRS should have to make one. A random password generator could also be used to make a default password for the superuser that is unlikely to be the same for a different installation of OpenMRS. Either way, the point is to make each installation's passwords different so an attacker can't just try the default across different installations successfully.

Adjust OpenMRS to use updated versions of Java

Using an outdated version of Java isn't just inconvenient for the users, it is also a security liability. If you use an older version of software chances are there have been vulnerabilities found that many attackers could exploit. Then there's the inconvenience of having to install the older version of Java just to get OpenMRS to work. I would recommend they adjust the system to support the use of the newest version of Java. This will bolster the security vulnerabilities left by the older version, and simplify the installation progress making it easier for users to use OpenMRS.

Update Wiki

The wiki still needs more information because certain areas are lacking and we weren't able to find what kind of implementations they had in place. When we researched the wiki for how they are handling some of the design principals we weren't able to find anything that was applying to the principle we were looking for. For example when we researched how OpenMRS

implemented Isolation, Open Design, and Economy of Mechanism we weren't able to find much information pertaining to these principles and how they are keeping these sections secure. Some of these principles had tickets opened up and there were complaints already in place.

Password Requirements

As of right now there are no password requirement when creating the root account, the user could easily set it to "password" if they would like. There should be a Template of what the password would need for the account. For example a good Idea would be for an account the password must be more that 12 characters long, have at least 2 unique characters in it (ex. !@#$%) and have at least one capital, and cannot contain an English Dictionary word or name. This would greatly increase the security of the database and the lower the chance that someone could brute force the passwords to gain access.

Conclusion

Write a one or two paragraph conclusion. (2 points per useful paragraph)

In conclusion, there are certainly changes that can be made to improve OpenMRS. The first thing that should be handled is the initial log-in credentials. It's not safe at all the have the username and password as admin and admin123. Initially this could be okay, but you should have something implemented that forces users to come up with a different username and password after the first time they log-in. Another way this could be handled is by requiring a new username and password be created right at the beginning. This way they will be changed for sure. It makes it far too simple for attackers to try admin and admin123 because it's very likely that users will forget to change or just neglect to change these credentials. That would be a relatively simple fix that would bring a significant increase to the security of OpenMRS and the database.

Another change that could be implemented into OpenMRS is updating their version of Java to being the most current version. With the version they have implemented it makes a large inconvenience for the user to have to go out of their way to downgrade their version of java just to be able to get openMRS to work. Also as stated before using an older version is easier to exploit and many attac1kers will find a way to get in easier. In short, updating their system will help their security and it will be more convenient for any of the users that want to use OpenMRS.

OpenMRS Database Audit

May, 2015

Authors:

Mouctar Diallo (OpenMRS ID mdiald), Richard Pinter (OpenMRS ID rpint001), Alex Torres (OpenMRS ID 10LeftFeet), Kyle Williams (OpenMRS ID region39)

Executive Summary:

OpenMRS is a medical record system with a great deal of security vulnerabilities. While the developers behind OpenMRS have done a great job at putting together a useful piece of open source software, there are several significant security problems that exist within OpenMRS. For this reason, OpenMRS is used primarily in third world countries. The software has far too many security risks to be considered for use in countries where things like patient privacy and computer security are held to such a high standard. This is especially true for the medical field.

The database structure that OpenMRS employs has several significant security flaws. The database files are stored unencrypted, all together in their own location. Anyone who is able to log in to OpenMRS has access to all the information in the database. Unless special permissions are enforced, OpenMRS has no way to enforce who is allowed to read or write to the database. OpenMRS also has no implemented auditing capability to speak of. This means that there are no audit logs to look at in case somethings goes wrong.

There is no doubt that the team behind OpenMRS has created a very useful product. Its open source nature allows it to be used by anyone, and anyone who wants to contribute to the project and make it better can do so. However, if those involved with the creation and widespread use of OpenMRS want their software to continue to grow in its use across the world, they need to take a step back and put a lot more focus into stepping up security.

Scope:

The OpenMRS database stores information about patients and their medical caretakers. The OpenMRS development team use Java and MySQL to keep their database running. The Database Audit team will be looking at how OpenMRS keeps track of activity that occurs relating to the database as well as whether or not it keeps track of who accessed the database and what changes they made.

OpenMRS Installation:

Operating System	OpenMRS Download and Installation Experience
Windows	Installation of OpenMRS on Windows was a bit of a challenge but was certainly do-able. First I downloaded the standalone version of OpenMRS from the OpenMRS website at http://openmrs.org/download/. Make sure it is the most recent version, which is at this time, OpenMRS 2.2. This will download a

Operating System	OpenMRS Download and Installation Experience
	compressed folder containing all the OpenMRS elements. Extract this folder. As stated in the readme file, to start OpenMRS, run the openmrs-standalone.jar file. This is where the major problem was encountered. Using modern software, this jar file will not do anything. We had learned that OpenMRS requires version 6 of java to run properly, which is a few versions behind the current version. Using this outdated version of java could potentially be a big security risk. It took me a while to find the right version of java 6, but once I did, OpenMRS was able to start up. It took about a minute for the software to set up all the necessary file structures. Once this was complete, it automatically opened a page in the web browser prompting a login. The readme provides the default username and password and strongly suggests that the user should change the password before doing anything else. However nothing really prevents the user from not doing so. I also found the process to change the password somewhat difficult. There were many fields to be filled out which I did not do correctly the first time.
Mac	Installation on a mac is fairly intuitive and simple (on mac version 10.10.1). First, install MySQL from the MySQL site. Then install the openmrs-standalone-2.2 zip file from the openmrs download page. Follow all the installation prompts provided. Open the openmrs file, and click on openmrs-standalone.jar. If it doesn't run, follow the prompt; if your security settings are blocking it, go to system settings and allow access for it. If you don't have it already, you will be asked to download a jdk file. Click on more on the prompt; it'll take you to oracle's site to download the jdk file. Accept the license agreement and download the jdk for Java SE Development Kit 8u45 for Mac OS (jdk-8u45-macosx-x64.dmg). Clicking openmrs-standalone.jar should now open a prompt, asking you to choose Demonstration mode or Starter implementation. Select Demonstration mode for testing purposes. The application should now open, creating a database with "test" patients. *Correction, you will need to downgrade to java 6 for Mac as well, as with java 8 you will eventually get an error trying to connect to the server. For now, instead of installing jdk-8u45-macosx-x64.dmg, download the java 6 sdk via apple.
Linux	I installed OpenMRS using Linux Mint 17. Installing OpenMRS on Linux was very tedious and probably wouldn't be easily done by someone without a basic understanding of computer file structure and the HTML language. Luckily, the download and installation of OpenMRS is likely going to be taken care of by an IT professional. I didn't have any trouble finding the link to download OpenMRS. I went to http://openmrs.org/ and clicked “Get OpenMRS Free” under the Download tab on

the top right-hand side of the screen. I found it peculiar that the option said “Get OpenMRS Free”. Users might wonder if there is a paid version once they see that. Once I clicked “Get OpenMRS Free”, I was brought to the web page from where I would initiate the download. I clicked the big orange button labeled - “Download” and was redirected to Sourceforge's website. This was a bit unsettling because I didn't expect to be redirected. At first, I wondered whether or not I inadvertently clicked an advertisement. Upon further inspection of the web page where the big orange “Download” button was, I noticed that the download description said “267.0 MB at sourceforge.net”. Stating where the download will come from in the download description is a good idea, but I also think it would be nice for the user to have a warning before being redirected. The ideal option would be to host the download entirely on the OpenMRS website and abandon the need for a third party to provide the download.

The download finished in a very reasonable amount of time considering the functionality that OpenMRS provides. I unzipped the file and soon realized that it was not evident how to start the installation process. I found myself being forced to read the README file. I also had to make the “run-on-unix.sh” file executable before I could run it. An installation should be very easy and should not require the user to read a README file nor enter commands into a terminal. At most, the user should only need to extract the file, enter the first folder, and click the installer icon. Everything else should be handled within a GUI.

While the installation was running, I had two options for the type of setup that I wanted. Unfortunately, I couldn't read the full description of the "Demonstration Mode" setup.

Once the installation was finished, I saw that a dialog box was opened and that it was responsible for allowing me to connect to the OpenMRS server. I shouldn't need to manage a dialog box that keeps me connected to the server. My computer should maintain connection to the server so long as I'm sending requests to the server at regular intervals. I recommend implementing a timeout feature here. The user would be disconnected from the server after a specific amount of time with no activity.

I tried logging in to my account using the web page that popped up after the installation finished. Fortunately, I read the README file and knew my default username and password. The username and password should be given to the user in a GUI during the installation process in addition to being put inside the README file. After typing the username and password into the fields provided, I tried to click the “Login” button. The button was disabled, and it would not submit my username and password. First, I tried clicking - “Can't Log In?”. It brought up a dialog box that said “Please contact your administrator”. This wasn't very helpful

to me since I am the administrator. I had to open Developer Tools and change the paragraph element that was responsible for the button. I deleted the "class" section of the p tag related to the “Login” button. Once I did this, the login button was no longer disabled, and I was free to access my standalone version of OpenMRS.

The process of downloading and installing OpenMRS is manageable for an IT professional. However, it is unlikely that the average user would be able to successfully complete this process themselves. Once again, I would suggest that most of the installation process be done through a GUI.

Assets:

Name of Asset	Type	Class	Value	Description of Asset	Threat Agents	Threats
Username & Password	Data	SEC	Moderate	The username and password are important assets to keep safe. For example, if someone has access to the admin's username and password, they could access anything or change whatever they want.	• Employees of the medical clinic • Terrorists, Government Rebels	• **Possible*Moderate -** Two employees could become aggravated by each other. If one of them has access to the username and password of the other employee, he/she could change or delete important information using the employee's credentials in order to get that person fired. • **Possible*Major -** An employee could get fired or laid off and decide to take confidential information with him out of spite for the medical clinic. • **Rare*Major -** A terrorist/rebel could demand the username and password of a medical clinic employee and then use them to

Name of Asset	Type	Class	Value	Description of Asset	Threat Agents	Threats
						access information within the medical clinic.
Database Files	Data	SEC	Major	The files that compose the database are some of the most important files in the system. These files are where information is stored about the patients and all of their medical information.	• Employees of the medical clinic • Personnel with remote access	• **Almost Certain*Catastrophic -** These files seem to be stored plainly on the system, and are not hidden in any way. Each of these files also has a name that states explicitly what it contains. This may be good for IT specialists or others who work with the system, but it could lead any of these threat agents to the exact information that they want. • **Likely*Catastrophic -** It also appears that these files are not encrypted. This means that any threat agent that does make it to these files would be able to read them assuming they had the appropriate SQL software. Most SQL software is open source, so this would likely not be a problem for them.
MySQL	Software	Other	Critical	This is the most popular database	• Programmers working on software using MySQL • Person	• **Likely*Catastrophic -** MySQL only includes auditing in the commercial version. Due to the high costs,

Name of Asset	Type	Class	Value	Description of Asset	Threat Agents	Threats
				software in use today, both with a free and commercial version available.	installing MySQL	many would opt to go with the free version. Since there's currently no database auditing tools available with OpenMRS, it's highly likely that users will have no auditing whatsoever. The risk can be less severe depending on the context in which the software's used. • **Possible*Major -** It's possible for programmers to write code that's vulnerable to SQL injection attacks. The risk can be less severe depending on the context the software's used. • **Possible*Major -** It's possible for programmers to write code that's vulnerable to Buffer Overflow attacks. The risk can be less severe depending on the context the software's used.
System Privileges	Data	SEC	Critical	This can include privileges to assign roles, create columns, create procedures,	• Hackers • Database Admin	• **Possible*Major -** A lot of Administrators unfortunately never change the passwords granted upon installation - allowing attackers to very easily log in and gain DBA (Database

Name of Asset	Type	Class	Value	Description of Asset	Threat Agents	Threats
				etc.		Administrator) privileges. • **Likely*Major -** It's common for Administrators to give more privileges to roles and users than necessary. These can be used as vectors of attack for attackers (sometimes even outright abused by the users themselves).
Patient Address	Data	PII	Moderate	The address of a patient is the location that they consider to be their home.	• Advertising Agencies	• **Possible*Moderate -** Administrative personnel might accidentally throw out a document with a patient's address on it.
Patient Phone Number	Data	PII	Moderate	The phone number is a phone number through which the patient can be contacted. It doesn't necessarily have to be the patient's personal phone number.	• Telemarketers	• **Possible*Minor -** A person posing as a patient (in person or over the phone) could ask the front desk to verify that the medical clinic does indeed have the correct phone number by asking the employee to recite the phone number that is currently on record.

Name of Asset	Type	Class	Value	Description of Asset	Threat Agents	Threats
Patient Name	Data	PII	Moderate	The name of a patient should be treated as confidential.	• Medical clinic employees	• **Unlikely*Moderate -** A staff person could accidentally call the wrong number and ask for a patient by name. The person on the other line would then know that a person with that name is a patient at the medical clinic.
Allergies	Data	PHI	Moderate	Allergies can be fatal to a patient.	• Angry Spouse	• **Rare*Minor -** A patient's spouse could use the patient's allergy to do harm to the patient. A spouse can acquire this information by talking to the medical caretakers on behalf of the patient.
Diagnoses	Data	PHI	Moderate	The diagnosis of a patient is private due to the possible humiliation some illness bring.	• Death Services	• **Unlikely*Insignificant -** A medical caretaker could accidentally say the medical condition of a terminally ill patient during casual conversation.
Weight	Data	PHI	Moderate	Some patients are very self conscious about how much they weigh.	• Fat loss companies	• **Rare*Minor -** Thieves could steal paperwork containing information about the weight of patients and sell it the fat loss companies.

Risks:

Extreme Risk

Potential Lack of Auditing

Since there is currently no system for database auditing in OpenMRS, there's a high chance there will not be any form of auditing implemented at all. Due to this, fact, it's safe to assume the following crucial controls aren't being implemented, controls that are very helpful in repelling a wide variety of attacks (such as SQL injection attacks, buffer Overflow, Social Engineering, etc.):

1) Since there's no auditing, there are no audit logs to check. Not only should audit logs exist, but they should be monitored, examined, and maintained; doing so helps detect and recover from attacks. Without audit logs, attacks will go unnoticed.

2) Without auditing, there are no logs to help keep track of authorized and unauthorized devices. There's a very good chance employees will bring a device that's compromised to work (such as phones).

3) Without proper audit logs, it's not possible to properly conduct penetration tests and exercises.

4) Likewise, without auditing one cannot properly respond to incidents in a timely fashion. Any procedures and protocols that may be in place would be rendered useless since it'll most likely be too late, if there are even any in place (highly unlikely since procedures are built upon a means to discover threats).

5) Even if you have some forms of data protection methods in place (such as encryption), without audit logs there often wont be any way to know if data has actually left the system; thus, data protection controls would be lacking. It should be logged when data is used, moved, or stored.

6) With a lack of auditing, there's no means to properly monitor accounts. There'd be no way to tell if, say, an attacker discovers and exploits an inactive account. It'd actually even be difficult to locate and disable inactive accounts to begin with.

Considering that having auditing is a prerequisite for many of these controls (as well as many others I haven't mentioned), I'm highly confident that none of these would be implemented (many of which are standard security practices).

Database Admins, and Granting Too Many Privileges

Making sure attackers do not get DBA privileges is always a concern; likewise, you don't want information to be leaked or unwanted access to occur from too many privileges being granted. These problems are, of course, also exacerbated by the lack of auditing. Some of the crucial controls lacking are as follows:

1) There should be processes and tools in place to identify and prevent the unwanted use or assignment of privileges (especially administrative ones), as well as the means to correct and control these actions should they occur.

2) There should be processes and tools in place to identify and prevent secure access to important assets, as well as the means to correct and control this access should the need to arise, based on who/what should have access to said information. Auditing is crucial in detecting access. Moreover, this is a common problem with OpenMRS as all too often, too many privileges are granted.

OpenMRS does have certain methods in place to aid in this, by utilizing roles, information and by raising awareness(documents and manuals). Once auditing is implemented, access to audit logs is another set of privileges that should be thought of.

Design Principles:

Principle	Definition	OpenMRS Grade	Reasoning
Economy of Mechanism	This principle requires that the OpenMRS database is user friendly, so that all users can use it easily without having a hard time understanding it. Moreover, economy of mechanism implies that the OpenMRS database audit is easy to implement and use. The audit functions that are already implemented are very straight forward and easy to understand. However, there is a very limited number of these functions.	E	
Fail-safe Defaults	If the database suddenly fails to function properly, the audit file should be safe and secure. This principle is very important. OpenMRS already has a few fail-safe defaults implemented within what little database auditing they do have.	E	
Complete Mediation	Implementing complete mediation would mean structuring the design such that before any user can have	E	

Principle	Definition	OpenMRS Grade	Reasoning
	access to any asset in the system, they must pass through some security steps. OpenMRS already requires users to enter a username and password before giving them access to the assets for which the have permissions. Unfortunately, the audit file is only keeping track of the person who made changes. It would be better if the audit could keep track of who is logged in, when they logged in and logged out, as well as what each person did while they were logged in.		
Open Design	This means that all aspects of the project's design are available to the general public so that they may be observed, scrutinized or improved upon. This includes aspects of security as well. Open source projects almost always have open design in mind.	A	Open design has some explicit advantages and disadvantages. Open design can be advantageous in that having your system open to the public allows for people to provide criticism and offer ways to improve the system. However it can also be very disadvantageous in that anyone who has malicious intent toward the system could potentially learn about all of its inner workings. This could end up being a security risk. In terms of open design, OpenMRS stays true to what open source projects are all about. Because of its open source nature, people from all around the world are able to help improve OpenMRS.
Separation of privilege	This means that because each group or individual is given the least amount of privileges as possible. Therefore, it would take more than one person to access important or	E	Unfortunately, because OpenMRS does not have a very strong least privilege, their separation of privilege also suffers. It would appear that every user that is able

Principle	Definition	OpenMRS Grade	Reasoning
	critical parts of the system. This strategy is employed so that if just one person in an organization has malicious intent, they won't be able to carry out any security breaches without cooperation from others with the remaining necessary privileges.		to log into OpenMRS has the same privileges to do anything that everyone else does. There may likely be a way to change the privileges of certain users using some other means, but such a feature does not appear to be a part of OpenMRS.
Least Privilege	This means that every user or program should have the least amount of privileges necessary to get their job done. If any user has more privileges than they need to complete their tasks, then they likely have enough privileges to cause damage to the system if they have malicious intent. It is the concept of least privilege that gives separation of privilege its effectiveness.	E	Users in the system are likely to have different privileges. However when logged into OpenMRS, it would seem that any user is capable of accessing the database by conventional means. Some users will certainly need access to the database, but if anybody who can log into the system has access to the database, this could lead to a major breach of security and privacy in the system.
Encapsulation	Encapsulation is a form of isolation. It focuses on an object-oriented approach. Using encapsulation means that data objects should only be accessible through procedures. This type of design helps to mitigate accidental and purposeful attempts to irresponsibly change a data object.	E	
Modularity	Modularity is a security design principle that focuses on the development of security functions as separate protected modules. Using modularity means providing common security functions and services throughout the design of your program. The primary reason	E	

Principle	Definition	OpenMRS Grade	Reasoning
	for giving consideration to modularity is that individual parts of the security design can be easily upgraded without the need to restructure the entire design.		
Layering	The process of layering encompasses providing multiple layers of security to your design. Providing multiple security layers is a technique known as “Defense in Depth”. Defense in Depth is a useful technique to use throughout your design because it has the potential to lessen the damage taken from any one security breach.	E	There is no database auditing in place. Therefore it's impossible to implement layering regarding the implementation records. Once database auditing does get implemented, I would recommend that activity records are segregated based on the time frame in which they occurred.
Least Astonishment	Least astonishment is a design principle that really concentrates on a users reaction to your security implementations. Your security implementations should make sense to a user, and they should understand the necessity for your design. The user doesn't necessarily need to know the reason for every detail of your design, but your security implementations shouldn't astonish the user. If a user doesn't understand why a security implementation is necessary, they will likely try to circumvent your attempts to provide better security. In essence, your design is more secure when the user understands the need for it.	E	I recommend that the database audit records be as conservative as possible. It is important that the user understands the necessity for auditing. A user will better understand the necessity for auditing if they feel that the amount and type of auditing is not overly intrusive. It is good to monitor user activity, but auditing that is done unnecessarily will likely aggravate users.

Summary of findings:

Any risks that database auditing would help prevent are not adequately controlled since there is no form of database auditing. As a result, nearly all of the design principles were violated (earning a score of E). The only passing score was for Open Design (which earned an A) due to the open-source nature of openMRS.

Recommendations:

Name of Recommendation	Recommendation
Develop Database Auditing	Before anything else can be done, a system of database auditing has to be developed. I'd recommend a system based on stored procedures. Any time a database table that requires auditing has a certain operation performed on it, or an auditable action occurs, call a generic procedure. The following should be considered when developing database auditing: Audit logs should include a date, timestamp, and, if data is moved, destination, as well as any tables that were accessed and any actions that took place. Ensuring the accuracy of the time listed for the logs is highly important. A possible way to aid in this could be utilizing two synchronized time sources (like Network Time Protocol) to keep log time stamps accurate. Set them all to the time zone (like UTC). There needs to be enough space to store the audit logs. Audit logs often need to be archived for extended periods of time. The logs should be stored in their own separate system. This should be mentioned in setup manuals, documentation, etc. There should be a way to provide for automatic reporting on disabled accounts, inactive accounts, and newly created accounts. Having a system to help keep inventory will help auditors more effectively monitor suspicious activity (i.e. attempted access of disabled accounts).
Curtailing the Abuse of Privileges	As mentioned, too many privileges are often granted. Each user should only have the minimum privileges they absolutely need to do their job. Users should also have their privileges separated so that a single individual does not have too much power within the system. This information should definitely be mentioned in any training manuals. The use of administrator privileges needs to be audited. Likewise, it should be mentioned that these audit logs should be monitored for unusual behavior. Assignment of new privileges should also be audited. These logs are useful for monitoring escalation of privileges.
Unencrypted Files	It is highly recommended to hide the most important information on a database - for example, the files that hold the patient's information and the

Name of Recommendation	Recommendation
	user's information. Using some type of encryption will make it hard for threat agents to get access to it. Moreover, the important assets should be separated into different files. This will allow for one asset to be accessed without the others being harmed.
Terrorist Attack	OpenMRS is used in third world countries. Most of these countries have been or are still going through some type of rebellion or series of terrorist attacks. Therefore having good security is very important. A good accounting log should be implemented. Furthermore, it is important to have the files save in servers that are located in different places. This allows for one infrastructure to be under attack because the files are still safe and functional on a different server.
Insider Threats	OpenMRS needs to consider informing their users about the risk of insider threats. Most businesses will indeed attempt prepare themselves, to some extent, for cyber related attacks and physical security breaches. Unfortunately, some companies forget to secure themselves from insider threats. It is imperative to consider security threats that originate entirely from your employees. Most insider threats originate from a lack of security awareness among employees. As a result, it is important to consider administering training to each employee who works with sensitive information in your company. This training will help to mitigate the security risks associated with employees handling sensitive data. The training should be made mandatory for all employees that are directly and indirectly handling sensitive information. The training should also be administered on a reoccurring basis. I recommend that the training be conducted monthly. It is critical to remind employees of the importance of security and how it can be practiced at their level. It is also important to keep employees up to date on the most recent kinds of social engineering attacks that are happening to other companies alike. The level of success of a defense against social engineering attacks is hard to assess. However, a company can require employees to report attempted and successful social engineering attacks. The success of training can then be measured based off of the level of knowledge and understanding, regarding an attack, that is displayed by an employee in their report. An employee with a good understanding of security is going to be much more aware of the fact that a social engineering attack is being conducted on them, and they are going to be more capable of providing a thorough report.

Conclusion

In conclusion, the OpenMRS database stores a lot of information, including medical records and user's and patient's information. Installing OpenMRS was a challenge in the beginning, but we were able to have it in three different operating systems (Windows, Mac, and Linux). After installing the database, we were able to identify important assets and categorize them into different types and classes. Moreover, we took the assets and pointed out the likelihood that those assets would be attacked, and if they were attacked what would be the cost of the damage.

The OpenMRS database does not have a lot of audit going on, there are only few auditing functions implemented, but they are not being used. The OpenMRS database already has some design principles in place, such as Open Design and Fail-Safe faults. On the other hand, the database is missing some of the design principles, like Separation of Privilege, Least Privilege, Layering and Least Astonishment.

Database Confidentiality May 2016

May, 2015

Authors

Pat C., Dan E., Jared F., Kevin G.

Executive Summary

We discovered that installing openMRS on a Windows Machine proved to be more difficult than originally thought. It was discovered that Java 6 was needed, so we had to downgrade the machine. This creates security issues that are addressed in this document. OpenMRS needs to be updated so it can support the newest version of Java to keep it secure.

We also found many issues when it came to the power given to the Administrator. If they do not change the default password and username it could make it very easy for an attacker to gain access to the system by using the default password and username. They are also in charge of assigning privileges and credentials to users, and if this is not done properly users could access information that they shouldn't be able to.

If OpenMRS could come up with a simpler way to code their database it would help them to improve security because if a mechanism is kept simple, it would be much easier to follow and fix along the way. The OpenMRS documentation is strong but doesn't define a standard way to improve security for the database. There should be a section for security setups of others so that people can sort of model a more secure system.

Scope

We will be assessing the database portion of OpenMRS. This database is the backbone of the system because it is where all of the patient information is documented, such as; name, age, gender and health status. The information stored in this database is not only personal but it is also valuable and needs to be secure. We will be assessing how well the information is protected from people trying to find it or attack it. If this information is not properly protected people may be at risk of identity theft as well as having their information sold to companies or other people. In order to assess the confidentiality of the system we will have to assess the value of the information stored in the database as well as the likelihood it will be attacked. Based on those assessments we can determine whether the information is secure enough.

Installing OpenMRS

We found it difficult to find a download available for just the basic openMRS data base. We found a Standalone Edition under openmrs.org/download that explained that the standalone edition just includes an embedded datebase and application server, so we would have closest access to look at the database for confidentiality issues. We downloaded to 1.11.2 WAR file and unzipped it but could not get the .js file to execute. When attempting to open the .js file using

Java Platform SE binary nothing happens. We also attempted to execute it using the Windows Command Line but we could not get it to work.

We now realized that the standalone addition was the most basic we could get so we downloaded the standalone platform from the Platform Release Notes 1.10.1 for standalone openMRS.
Where to go for information about the platform release
https://wiki.openmrs.org/display/RES/Platform+Release+Notes+1.10.1
The link to download the standalone version of openMRS.
http://sourceforge.net/projects/openmrs/files/releases/OpenMRS_Platform_1.10.1/
We download the standalone 1.10.1 and ran the Executable Jar File and it just opened a blank browser window with the address http://localhost:8081/openmrs/initialsetup

We also attempted to download openMRS standalone 2.2 to see if there was any difference between the updates. When running it using the jre7 it opened a window in the web browser saying OpenMRS is un able to start because; ClassReader failed to parse class file – probably due to a new Java class file version that isn't supported yet: class path resource. We have not attempted to get this working.

To see if it was an issue with the java on the computer we were using we copied both standalone folders and the .jar files we download in an attempt to get standalone openMRS to work and I had the same problems.

We have since discovered if you install the java jre6 on your computer and download openmrs-standalone-2.2.zip, unzip the folder and run the executable jar file it should run on Windows Machines.

Assets

Patient's Name and Unique Identifier in the Database

Type: Data Asset

Class: PII (Personal Identifiable Information)

Value: Moderate

Description: Each patient's name is stored in the database and is linked to all their records that are on file. Patient are able to be searched by name through the database.

Threat Agents:

- Criminal Exploitation
- Stalking
- Identity Theft

Threats:

- **Likely*Major:** An attacker uses the information to further identify and learn about a specific person's information and gain access to all the records that are on file.
- **Possible*Moderate:**Stalkers can obtain the information about their victims and use it for their own personal gain or sick satisfaction.
- **Possible*Major:**An attacker gains access to the address of their potential victim or worse, get credit card numbers or social security numbers.

Patient Summary

Type: Data Asset

Class: PHI (Protected Health Information)

Value: Major

Description: An extended patient summary including, a summary of their last visit with date and time, a list of past visits with diagnosis and clinical notes, a list of known allergies, the patients vital signs from last visit which includes, heigh, weight, BMI, Temperature, pulse, respiratory rate, blood pressure, and blood oxygen saturation.

Threat Agents:

- Criminal Exploitation

Threats:

- **Likely*Major:** An attacker with access to PHI information could learn about any diseases a given person might have, any allergies, what kind of condition they were in when they last came in at a given date. They use this information to find the people that could be easily victimized and exploited.
- **Possible*Moderate:** An attacker accesses medical information and uses it to receive treatment or drugs that would normally go to the sick person. They could sell these drugs or even use them if that is their motive.
- **Possible*Major:** Political figures, state officials, or even wealthy or important business people are easily targeted by an attacker that knows they were registered in the system and gains their medical information.

MySQL Database

Type: Software Asset

Class: Other

Value: Major

Description: The heart of the actual database for the storage of all the information related to OpenMRS and it's clients.

Threat Agents:

- SQL injection attacks
- DoS attacks
- Insider Attack (company employee)

Threats:

- **Likely*Moderate:** Database is offline for an extended period of time due to city wide blackout.
- **Possible*Major:** Denial of Service attack done by a hacker.
- **Possible*Major:** An insider who ends up using the stolen information for revenge, profit, or even blackmail.
- **Possible*Major:** Through SQL Injection, the hacker tricks the system into spitting out protected information without the company knowing about it and become a hazard to private information of all the patients and clients of a given organization.

Work Stations (Data Entry and Point of Care)

Type: Hardware Asset

Class: Other

Value: Major

Description: The work stations used by the Data Entry clerks as well as the Point of Care physicians. These work stations will have a steady flow of important information going in and out of them everyday.

Threat Agents:

- Theft of Computers
- Environmental Disasters
- Inside Attackers (company employee)
- Viruses
- Key loggers
- Spyware
- Adware

Threats:

- **Likely*Major:** Virus' easily installed onto systems due to social engineering or plain negligence by users. Attackers use phishing, spyware, and adware to exploit systems.
- **Possible*Major:** Theft of computers, laptops, tablets, or even mobile devices that have information about patients can be tampered with by an attacker; They can change or delete important information that could result in harm for patients who require certain care.
- **Possible*Major:** Insider employee attacks targeting passwords and protected information for some type of personal gain.

Backup Servers for the Database

Type: Software Asset

Class: Other

Value: Major

Description: The servers put in place in order to keep all stored information safe incase of an attack or natural disaster. Without backup servers, if information of some kind is deleted, there is no way to recover it.

Threat Agents:

- Environmental Disasters
- Dos Attacks

Threats:

- **Likely*Moderate:** Database is offline for an extended period of time due to city wide blackout, before the back up generators can resolve the issue.
- **Possible*Major:** Attackers the take down or delete backup servers, could be doomsday for a company that hasn't saved the information any other way.
- **Possible*Major:** A power outage caused by a natural disaster like an earthquake, results the backup servers to be corrupt or destroyed.
- **Possible*Major:** The servers a physically stolen. This would be detrimental to a company that hasn't saved the information any other way.

Usernames and Passwords of Data Entry Clerks

Type: Data Asset

Class: SEC

Value: Major

Description: The individual usernames and passwords used by the Data Entry Employees when they go to sign into their work stations and begin to input and change data in the database.

Threat Agents:

- Phishing
- Brute Force
- Key logger
- Inside Attacks

Threats:

- **Likely*Moderate:** People choose to keep a standard password or don't try hard enough to protect it, which results in a hacker figuring out the login information.
- **Possible*Moderate:** Hackers get people to give up their passwords without them knowing by using phishing schemes through emails and pop ups.
- **Possible*Major:** Denial of Service attacks being launched as well as the the hacker changing, deleting or corrupting data along the way.
- **Possible*Major:** An attacker or insider utilizing key loggers or other backdoor entries.
- **Unlikely*Major:**Hacker skilled enough or has enough time to brute force their way into the system.
- **Unlikely*Major:** an Administrator writes down his username and password and the threat agent finds it.

Infrastructure Requirements

Type: Communication

Class: Other

Value: Major

Description: The requirements that must be met regarding, electrical systems, surge protection, voltage stabilization, security, staffing, construction, and server installation.

Threat Agents:

- Natural disasters
- Insider attacks
- Poor maintenance

Threats:

- **Unlikely*Minor:** A natural disaster such as a storm could cause a power outage that would cut off most of the communication and could hinder work from getting done. You would hope the company has generators and surge protectors to protect the system from data loss. .
- **Possible*Moderate:** One of the IT/Infrastructure professionals that is responsible for the upkeep and maintenance of the infrastructure could have been cutting corners on server updates causing the infrastructure to have vulnerabilities.

Administrative Privileges

Type: Data Asset

Class: SEC

Value: Critical

Description: Administrative log on with more rights than a normal user. Will have access to entire database and oversee the entire operation.

Threat Agents:

- Insider Attacks
- Key loggers
- DoS attacks

Threats:

- **Possible*Catastrophic:** A system administrators computer could be infected with a variety of different types of malware causing the computer to become a zombie, allowing the hacker to make changes to anything from user privileges to changing critical parts of the database from the backend.
- **Unlikely*Catastrophic:** An employee with a vengeance could take it upon himself to install a key logger on his computer and prompt the admin to come put in his login information to install something on his work station. This employee could then leak tons of information that he shouldn't have access to and make a huge breach of confidentiality.
- **Possible*Catastrophic:** The system administrator does not change to default username and password when they create the database. It is easy for an attacker to guess the known defaults and get access to the system with Administrator Access.

Billing Module

Type: Data Asset

Class: SEC

Value: Major

Description: Billing system that can be connected through different departments in a hospital, or a clerk from trusted pharmacies. It generates receipts of orders and bills of payment. This could contain bank information, credit cards, insurance information and very important information.

Threat Agents:

- Malpractice Theft
- Identity theft

Threats:

- **Unlikely*Catastrophic:** A hacker could utilize a method of injection on the billing module causing it to unrightfully spit out information that could contain credit card numbers or insurance information, costing the company a lot of money. A huge breach of the confidentiality of patients. The hacker could then post this sensitive information on the internet for his own personal gain.

- **Unlikely*Catastrophic:** An inside attack could lead to the leaking of many Social Security Numbers from all the patient's insurance information, leading to numerous stolen identities as well as lawsuits for the company breached.

Patient's Address and Phone Number

Type: Data Asset

Class: PII (Personal Identifiable Information)

Value: Major

Description: This is the personal address and phone number that is on file for each of the patients.

Threat Agents:

- Stalkers
- Identity theft

Threats:

- **Unlikely*Moderate:** the information could be removed and it would be difficult to contact the patient.
- **Likely*Moderate:** A hardware or software malfunction causes the information to be lost.
- **Possible*Minor:**spam mail is sent and spam calls are made to the patient.
- **Possible*Major:** The information is sold to sent junk mail to the patient.
- **Possible*Major:**Someone obtains patient information to make threats or go to their personal address.

Current Security

"Two Way Encryption: " Openmrs currently ustilizes the AES/CBC/PKCS5P adding method for block cipher encryption and decryption for the API to access the database

"Knowledge of SQL Vulnerabilities: " In the Openmrs documentation there is a page titled Top Vulnerabilities in Java Web Applications. This page list common security vulnerabilities in java code. This page is in the Openmrs Wiki so we can assume that Openmrs was coded with these Vulnerabilities in mind, and such attacks would not allow a threat agent entrance to the system.

Risks

Extreme Risks

Administrator Credentials

When logging into openMRS for the first time, a default password is used. The install instructions say that it is important that you change the password, but there is nothing in place to force you to do so. The install instructions is the only form of documentation I could find warning users to change their passwords from the defaults. The Administrator does have power to change users credentials, but they must have the knowledge to do so.

When looking through the documentation for openMRS, I was able to find very little regarding how credentials should be assigned to users. All I could find was documentation saying make sure you give the user a privilege. This puts a lot of power in the hands of the person that initially created the openMRS project. To reduce the probability of an attack more checks should be put in place on the Administrator so proper credentials are assigned.

I am confident in this assignment. The Administrator is given a lot of power but there is very little documentation explaining how to use it. I looked through a lot of documentation and could find very little information about how user credentials should be assigned.

OpenMRS has done little to control this threat. There is nothing in place to force a new Administrator to change his username and password, so if they fail to do so a threat agent could have a very easy time getting access to the system. There also need to be better documentation in place for assigning credentials and privileges. I could not find much in my research and it could be very confusing for someone with little computer skills to assign them properly. Improper assignment of privileges could lead to users getting access to information they shouldn't, a huge security risk.

High Risks

Using an Outdated Version of Java

When installing openMRS Java 6 was required to make it work. The documentation from openMRS says the Java 6 is the minimum but Java 7 is recommended. I did not see anything currently in place to address this issue.

OpenMRS not working with the current version of Java is a security vulnerability. According the Oracle Critical security updates are being released every couple of months. If a user is required to use an older version of Java, there are more chances for a breach in security.

I am confident in this assignment. I did a lot of research on installing openMRS on a Windows Machine and many people are only able to get it to work on Java 6. There is nothing regarding this as an issue from openMRS but running on an outdated version of Java is a security risk.

OpenMRS has done little to control this threat. There code needs to be assessed and a fix needs to be found so OpenMRS can run on Java 7

Design Principles

Economy of Mechanism

This principle pertains to keeping the database and the use of the database as simple as possible so that if something were to go wrong, it would be easier to fix. If the mechanism isn't kept simple, it will make everything harder to understand, model, configure, implement, use and troubleshoot. Keeping it simple also tends to have less exploitable flaws and requires less maintenance, which in this case since OpenMRS is open-source, would be a plus. If it's built simpler from the beginning, when errors occur, it will be way easier to track and fix that it would be with a more complex mechanism. When it comes to the database, it has to be built well in order to hold and protect all the information, especially in OpenMRS' case because it holds a ton of important health information.

Grade C: When it comes to OpenMRS, the data that is stored in their database is not exactly simple. Their data model is fairly complex, but it kind of has to be because they are dealing with a wide range of health information. However, it is very well documented in the wiki and I am sure that anyone implementing a complex data model has documentation to go along with their specific data, but that falls on the user.

Fail-safe Defaults

Fail-safe defaults are pretty important when it comes to housing a database full of important information. The database is constantly being referenced to and appended to and maintaining the confidentiality during all this is a tough task. OpenMRS handles this task pretty well with good default permissions and many modules and checks put in place to uphold the confidentiality of the database. It has a system called privilege tag that monitors all the pages and when a page is opened it makes sure that the user has the right permissions to open that page. If they don't, the page is unable to be opened. Since OpenMRS is dealing with health information, it is important to have different tiers depending on how valuable the information is.

Grade: B+ : They currently use an out-of-the-box role-based security method for access control of the information. The access control is split up a number of different ways. Everyone accessing the database is either a User or an Employee and priveleges are separated accordingly. Employees are then split up again into Roles, depending on their position in the company. Some roles include Data Assistant, Data Manager ect. Roles are declared by an Administrator at the beginning and each time a new user/employee is entered into the system they are given a role specific to what they are going to work on and privileges to match. The roles determine whether they can only read content on the database, whether they can edit content and users or whether they have the privilege to add/delete content and users from the database. These roles are well documented in the wiki and I believe are good steps in places to uphold the confidentiality of patients in the database.

Complete Mediation

OpenMRS does a good job at upholding the principle of Complete Mediation when it come to the database and the confidentiality. It has a system called Privilege Tag that monitors all the pages and when a page is opened it makes sure that the user has the right permissions to open that page. If they don't, the page is unable to be opened. This is important because when it comes to important health information that needs to be protected, the system needs something in place to check each users privileges against what they are trying to access. Everyone that is setup to access the database is separated into roles and given proper permissions for the work they are going to be providing for the company.

Grade B: OpenMRS does a good job in this section as far as upholding confidentiality of their database. They have a good system in place for monitoring and recording the pages accessed by each user. Overall I believe they did a good job, however, surfing through some of the comments, a user of OpenMRS brought up a good point which is why I gave the system a B. He brought up that the access control and privilege tagging worked good for information that is stored in the database but doesn't work so good for databases that primarily store documents. He was questioning how to better implement access control when using the database this way. Overall, solid job.

Open Design

OpenMRS is fairly open about everything on the backend because it is an Open-Source program. They also do a very good job on documenting everything on their wiki page. Open-Source software is looking over by a wide range of people because it is basically put in place as a starting point of something and is meant to be open to further implementation. There are many modules that can be appended to the system as well that will take it to the next level.

Grade B: Being that the program is already open-sourced gives it a head start in this category. However, they do a good job documenting the entire program in every way so that anyone using it from a user to an administrator to a developer has the documentation needed to use and implement the system.

Separation of Privilege

When dealing with a database that houses important information, especially pertaining to health care, it is very important to have a strict separation of privileges. You don't want someone that is supposed to be able to view the database to be able to delete entries for example. This also helps uphold the confidentiality of the database because only certain people are supposed to have access to certain information.

Grade B: OpenMRS does a good job in this area in my opinion. They have many different levels of privileges, that can be edited by the Administrator and they have modules in place to monitor and uphold these restrictions. Out-of-the-box they have Users and Employees separated and employees are separated even further depending on their role in the company. This allows for

confidentiality to be upheld, while giving users and employees the access that they need and no more than that.

Least Privilege

When dealing with the confidentiality of a database, it is important to keep everyone strictly in check when it comes to privileges and make sure that they have just the right amount of privileges to accomplish their task. If a user temporarily needs more access, the administrator needs to make sure to rescind that access after they are done working.

Grade C-: OpenMRS has a good role-based system put in place to keep everyone in check, however, right out of the box, all the privileges and roles are determined by the Administrator. They are unclear what privileges are given by default but stress on the wiki that the Admin be on top of accurately assigning the correct privileges when adding users.

Least Common Mechanism

When dealing with this design you should minimize the function shared by different users. This provides mutual security. It makes it easier to verify if there are any security violations. Users should only be granted access to what they need and nothing more. This can be accomplished by proper implementation of privileges and credentials.

Grade C: There is no good method in place for Administrators to assign privileges and credentials to users. Many of the people that are going to be using this program will not have much computer knowledge, and while you can assign privileges and credentials for users, there is no documentation on how to do so. This could cause a user to have access to information in the database that they shouldn't, possibly exposing sensitive information.

Psychological Acceptability

Especially when dealing with a database with important information, you want to make sure that the security mechanisms in place are strong and uphold the confidentiality of patients within the system. However, you need to make sure that the mechanisms do not overly interfere with the work of others within the system. If security mechanisms start slowing down the work of others they are sometimes opted to turn off these measures in order to accomplish their work faster and this can lead to security vulnerabilities. You also have to make sure that any security measures that are not transparent to the user, make sense to the user. They have to understand why this information is being protected so heavily to avoid them trying to turn them off. When it comes to databases, there needs to be separated privileges and systems in place to check for those privileges but logins shouldn't be required too often.

Grade B-: In my opinion, OpenMRS does a pretty good job at balancing fairly strong security without hindering the entry or lookup of data within the database. OpenMRS has many different levels of access and from what it looks like, it monitors the access on the backend and the only time that it will hinder a user from working is if they aren't allowed access to a specific page. These security measures in my eyes are justified and should not be able to be turned off,

considering OpenMRS houses information that needs to be protected. From reading through the documentation, I see that there are many modules that can make the system more secure, and when those are being implemented there should be specific notice to make sure it doesn't slow down the entry and lookup of data too much.

Isolation

The isolation of different levels of information is huge when it comes to upholding the confidentiality of a database. There needs to be different tiers depending on the importance of the data. Some information within the database may not need to be protected as heavily as other information while other data may only need to be accessed by the system administrator. These levels need to be determined from the beginning and protected accordingly to prevent any breaches of patient confidentiality.

Grade B-: OpenMRS houses a wide variety of medical information that needs to have different levels of security. There is some information within the system that isn't as sensitive and that might need to be accessed by a non-employee, like a medical student for example. From what we can see, OpenMRS does a pretty good job of isolating sensitive information from other types of less sensitive information. The only person that has access to the entire database would be the admin and they have well defined tiers of information that is able to be accessed by different roles within the company.

Encapsulation

Modern databases do an excellent job of tightly coupling code as well as data in a way that is accessible as well as secure. It goes along the similar line of the isolation of the data but helps more on the backend, keeping everything in one place and making it easier to oversee the entire thing.

Grade B: The database holds all of the information that is required for access by the API. This means the the database must have good security because a lot of valuable information could be lost if the wrong person were to access it. The database is secure but there are ways to gain access to the database through security vulnerabilities in the API.

Layering

This principle refers to the user of multiple and overlapping protection. There are multiple levels of protection so if one were to fail, the system would not be left stranded. There are multiple levels of security, and if one is broken in to, the others layers would stay intact in order to protect parts of the system.

Grade B: The database is protected at a core level by what openMRS calls a "Black Box" but there is not much else in place to protect the information in the database. There should be better security on other elements of openMRS, including Administrative power. OpenMRS is built in layers and each layer needs to have its own protection.

Least Astonishment

This principle means that when a user is using a system, any response or action done by the system does not surprise the user. In context to this project, when a user is accessing the systems database they should not be confused by a certain result of an interaction they are having with the system. Each response a user gets from their input should be exactly what they were expecting the database to do.

Grade B: The database responds just as one would expect it to, it's functionality and output would leave a user with little confusion of surprise. User input would result in expecting system output, which is something OpenMRS has done well. Despite this some work may still need to be done to keep database information protected, even with it's current security.

Summary of Findings

Risks

There is too much potential for a security vulnerability left in the hands of the Administrator. When the system is first set up there is nothing in place forcing you to change the password from the default. If the username and password were left as default it would be very easy for an attacker to gain access to the system.

The Administrator also has power over setting privileges and credentials for the users. There is very little information on how to do this available, so it could prove to be very difficult for a first time user to assign them all correctly. A user with an incorrectly assigned credential could get access to info they shouldn't, a major security threat.

To install openMRS on a Windows machine you must use Java 6, which is outdated. Using an outdated java could cause security issues because java is constantly updating its security, and openMRS cannot use those updates.

Design Principles

If openMRS could come up with a simpler way to code their database it would help them to improve security because if a mechanism is kept simple, it is much easier to fix. When it comes to the Economy of Mechanism openMRS could do a little better. The database has a good system in place for monitoring what a user has done when entering information into the database but do not have a solid system in place for monitoring a database full of documents, this is something that needs to be improved.

OpenMRS has a good method in place for keeping everyone in check. Privileges are assigned when a user is created and this dictates what they can and cannot do in the database. There are good security measures in place to keep users to only be able to access information they have privilege to access, but there is one major flaw in this system. The privileges are set by the administrator, making human error a major security vulnerability. There is a lot of room left for error because people with little computer skills are assign users privileges and credentials. This

could cause channels created between users where they shouldn't be, allowing unwanted access to the system. There are many layers in the system and they all need to be protected from attacks in order to keep the database safe from attack.

When it comes to Isolation and encapsulation, OpenMRS has a solid method to make sure that everything is in one place, simple and relatively easy to understand. The database is protected pretty well from the Core and out to multiple layers and performs just as you think a database would. It has a variety of modules that the administrator can implement to further make the system better and stronger.

Recommendations

Force the First User to Change Username and Password

In order to remove the possibility of a new Administrator forgetting to change the username and password from the defaults they should be forced to change them when they log on for the first time. The user should be forced to change their username and password when they log it. If you log on to a new session of openMRS and are allowed to keep the default username and password this has not been implemented correctly.

Create Proper Documentation for Assigning Privileges

The Administrator has power over assigning all the users privileges. Someone with little computer skill could easily assign someone the wrong privilege. There needs to be documentation available about how privileges and credentials work in openMRS. Once the documents have been created Administrators can learn how to properly set up their privileges.

Update OpenMRS to run on Java 7

On Windows machines openMRS does not work on Java 7, you must downgrade to Java 6. The code should be analyzed and the bug causing this should be corrected. Even if it is a difficult fix running on an outdated version of java creates security risks. When this issue is fixed openMRS will be able to run using Java 7 as intended.

Conclusion

Databases play a very important role in the storing of data, some data that could be extremely sensitive, making them a regular target for malicious parties. This requires a strong plan and implementation of security to make sure there are no breaches and leaks of information. Being that OpenMRS is a medical records system, it houses a wide variety of information ranging from some general public knowledge all the way to sensitive data that would be catastrophic if exposed to the wrong people. Looking through the OpenMRS documentation and platform we were able to find a long list of assets that need to be protected to uphold the database confidentiality, some data more than others. Along with these assets, we were able to come up with a wide variety of threats that could put the confidentiality of these assets in danger. In order

to do a proper assessment of the database we had to install it and play around with it. This proved to be harder than expected, but with Java 6 installed we were able to do so.

Since OpenMRS is the home to such a wide variety of medical information, in this risk assessment it was very important to go through each one of the design principles to see what needs security. We crosschecked what we thought each principle should have to uphold database confidentiality with any information or mechanisms listed in the documentation. They do a good job with the separation of privileges, which is crucial especially if this is going to implemented within a large company. We found areas that were strong and we found areas that were not so strong. Granted, there are many modules that can be implemented but since the system is Open Source, that falls on the administrator to implement for their given situation. Overall, OpenMRS demonstrates a pretty good way to implement a medical records database for cheap, but needs to be monitored and tweaked depending on the given situation. They offer a wide range of modules to help mold the system to the situation it is thrown into but it can't be forgotten that it is Open Source and requires a lot of maintenance that falls on the Administrator unlike regular programs that undergo many updates by the developer.