INF226 Obligatory assignment

Håvar Eggereide and Syver Storm-Furru

2016-10-02

Contents

Chapter 1

Introduction

For our obligatory assignment we were tasked with analysing OpenMRS, a medical patient journal system. The analysis is performed both using static and dynamic code analysis tools (HPE Fortify and FindBugs for static analysis and OWASP ZAP for dynamic), as well as a thorough manual run-through of the installation process and software usage.

The goals of software testing It is now common practice for a developer to adhere to a software architect's plan for the development of the software components, and many utilizes unit-testing as a supporting tool for developing code without errors. So what is then the benefit of having a separate plan and procedure for system testing from the view of security?

It has been demonstrated with many grim examples during this course and in the curriculum texts, the importance of security and the heavy consequences to the whole enterprise if it is neglected. The issue is of such a big scope that approaching it as just another design or implementation issue fails to appreciate the complexity it presents.

The goal of testing the security as a separate process is to try to map as much as possible of the potential security issues and security considerations that has to be kept in mind walking thru the process of development. To accomplish this it is necessary to approach the software and the architecture from the point of view of a agent with malicious intent.

In the end it is most likely not possible to make the software completely immune to security faults. It's a natural part of security testing to find problems where a full repair might not be achievable, it is then also the responsibility of the security testing process to develop mitigation plans for these vulnerabilities.

So the goal of security testing is to find, repair and prepare for security vulnerabilities.

Static and dynamic analysis Static and dynamic analysis are methods for checking the software for vulnerabilities. They are both done using software tools that help the user analyze.

Static analysis is performed before any code is run, and checks the code base for vulnerable code that can be abused. It checks different runtime behaviours, both standard and non-standard, in order to find potential backdoors, weak code and other vulnerabilities. Dynamic analysis on the other hand is done while the application is running, and works a lot like how an actual attacker would target the application, performing such things as XSS and SQL incjection attacks, as well as monitoring the system runtime, for example performance and HTTP requests.[1]

[1] VeraCode. *Static vs. Dynamic testing.* `https://www.veracode.com/blog/2013/12/static-testing-vs-dynamic-testing`. Accessed: 05. Oct. 2016. 2013.

1.1 Software involved

1.1.1 OpenMRS

OpenMRS is, according to the website, the "world's leading open source enterprise electronic medical record system platform".[2] It is used in hospitals and medical facilities all over the world, for example in Nigeria, South Africa, India and the United States, and is supported by many different governments, NGOs, and both for- and non-profit organisations. The software has a stated goal of being usable with no programming knowledge, and to be a common platform for which medical informatics efforts in developing can be built.

Technical Specifications OpenMRS is a client-server platform, with a web front end. It is programmed in Java 7, using Tomcat 6 or 7 as the server framework, and MySQL 5.6 as the database backend. It also exposes a programming API to users, and is modular and extendable.

Usage OpenMRS is operated through a web browser, and is centered around a main dashboard that, with just the base installation, will give you access to patient journals, as well as showing currently active patients. The system also lets you categorize patients/other registered in different persons in different categories, easing the process of generating statical analyses. The system also lets your register and book future meetings with patients.

1.1.2 HPE Fortify

Fortify is a code security tool suite, developed by Hewlett-Packard Enterprise (HPE). It aims to "make application security a natural part of the new SDLC, enabling time to market by building security in".[3] It contains such tools as WebInspect, a dynamic code analysis tool, and the Fortify Static Code Analysis tool. Fortify is a proprietary solution, but is available with an academic license for free.

We ran the Audit Workbench version bundled with Fortify version 16.10.

Audit Workbench Audit Workbench is the tool used to organise the the output of HPE's static code analysis software contained in the Fortify package. It is a GUI application built on top of Eclipse, specially designed for organising and presenting output for the HPE tools.

The installation process for Audit Workbench was straight-forward, but running the program required changing variables for the Eclipse backend, without information about how this is done readily available. The software also was a large RAM consumer, needing 5 gigabytes of RAM to analyze a relatively large project (OpenMRS).

[2] OpenMRS. *About OpenMRS*. `http://openmrs.org/about`. Accessed: 02. Oct. 2016. 2016.

[3] HPE. *Application Security*. `http://www8.hp.com/lamerica_nsc_carib/en/software-solutions/application-security`. Accessed: 02. Oct. 2016. 2016.

1.1.3 FindBugs

FindBugs is a static analysis tool for scanning java code looking for potential errors in the implementation. It is distributed under Lesser GNU Public License. The project originated from the University of Maryland.[4] It runs on java 1.7 ether initiated from a terminal or you may use a simple GUI which makes it easier to get familiar with the program.

We ran FindBugs 3.0.1 for our tests.

1.1.4 OWASP ZAP

ZAP is a dynamic analysis tool developed by OWASP, the Open Web Security Project. The software acts as a proxy between the host computer and a web application, performing different types of automatic scans, as well as having tools for manual searches for security vulnerabilities.[5]

We ran ZAP version 2.5.0 for our tests.

1.1.5 ThreadSafe

"ThreadSafe is a static analysis tool for finding concurrency bugs and potential performance issues in Java programs."[6]

The program is developed by Contemplate ltd. aiming to help developers discover concurrency issues in the code-base prior to the deploying of the application. Concurrency is a complicated issue, having the correct understanding of what it is and how it works is not easy. Then the next step of implementing it correctly can give rise to more complicated issues which are even harder to discover. For the developer to have a tool to aid in working with concurrency is of high value. Should a issue arise during runtime the ramifications, for a system like OpenMRS which potentially need to be operating 24 hours a day, could possibly be disruption of the integrity of the database and halt of operations.

ThreadSafe is not open source and it is necessary to invest in a license if there is a need to integrate the application in the SDLC. There is however a free trial of 14 days available.

[4] University of Maryland. *FindBugs - Find Bugs in Java Programs.* 2015.
[5] OWASP. *OWASP ZAP 2.4 Getting Started Guide.* 2016.
[6] Contemplateltd. `http://www.contemplateltd.com/threadsafe`. Accessed 15. November 2016. 2016.

Chapter 2

Setup and Configuration

2.0.1 OpenMRS

The setup process of OpenMRS is quite involved and time consuming when attempting to do so on a personal computer, requiring both Tomcat, Java and MySQL to be setup. The official documentation is useful, but different parts of it uses different versions of e.g. Tomcat, so it can be confusing. It also provides install instructions for Windows and Linux distributions with Aptitude, but not for OSX or other Linux distributions.

As mentioned, OpenMRS doesn't run on the newest version of MySQL (at the time of writing MySQL 5.7), and the install instructions do not mention this. The process of figuring this out, and of removing and reinstalling a previous version of MySQL, proved to be a lengthy detour on an already long road. The instructions are also not very specific when noting which files you need to run OpenMRS in Tomcat, whether it is the source code, which was difficult to build and only return a test suite on a normal compile, a readily packaged complete install (which did not work properly), or a .war file that should be uploaded to the Tomcat server.

We first attempted setting up OpenMRS on OSX 10.11, but ran into problems when trying to install the correct version of MySQL, and therefore retried in an empty virtual machine running Linux (tested with both Lubuntu and Kali Linux). Following the install instructions were a lot easier when running Debian based distributions containing Aptitude, but we still had to find and install a previous version of MySQL.

Once everything was installed OpenMRS had some extra setup that was required, done through a web interface. This was mostly easy once the correct version of MySQL was in place. You were also given a first username and password that was, respectively, 'admin' and 'Admin123'.

2.0.2 Tools

Audit Workbench Audit Workbench was set up by downloading the HPE Fortify software suite, which contains installers for different Fortify software. The installer for Audit Workbench was run, without any issues. Setting up the Audit Workbench IDE required changing the -Xmx setting in a deeply nested "eclipse.ini" file, which required a bit of dedication to find. The reason for this is that scanning a project requires more memory than is allocated to Audit Workbench by default. After that, running Audit Workbench was pretty straight forward.

FindBugs The stand-alone application which is distributed from the projects main page was easy to get started from the terminal(OSX-bash) without any setup other than the first installation. Starting the initial scan was also straight forward. The high number of errors reported was misleading and proved a cumbersome task. What proved to be a better solution was using a FindBug-plugin for the IntelliJ IDEA. This way IntelliJ made a build of the code base and then ran the scan on this build. IntelliJ could reason about the classes which where not necessary to scan, as per example all the unit tests. The code base we used for the scan was cloned directly from github (https://github.com/openmrs/openmrs-core).

OWASP Zap Running ZAP requires ZAP to be installed in the same location as OpenMRS when running OpenMRS locally. Therefore, we set up a virtual machine to run both ZAP, the Tomcat server and the MySQL server. The virtual machine used ran the the linux distribution Kali Linux, which is a specialized distro for penetration testing. It included OWASP ZAP, and we therefore did not need to set up the application itself. However, fully utilizing ZAP required using an addon for our browser (Firefox). The addon had an out-of-date signature, and thus needed Firefox to run addons without valid signatures. After changing that setting for Firefox and running the setup wizard for ZAP inside Firefox, it was ready to be used.

ThreadSafe ThreadSafe has three install options, two which are available in the trial version. These options are: A .jar file which is executable from the terminal taking input-arguments and a setup file for each project, plugin for a static analyzing tool called SonarQube and a plugin for the Eclipse IDE.

As we were using a free trial we had to be limited to ether using the terminal program or the Eclipse plugin, which were the included options in the trial. As setting up a project configuration file for the terminal application seemed to open the possibility of misconfiguration, using the Eclipse plugin seemed the best alternative. Installing the plugin from the .zip file was easy, the only other thing necessary was to order a trial license and paste this license in the Preferences subcategory ThreadSafe. We then let Eclipse build the Maven .pom file from the OpenMRS source-code and everything was ready for scanning.

2.1 Preliminary results

2.1.1 Visible potential issues

During the install process we were on the lookout for potential vulnerabilities that were visible to us. Two things stood out: 1) The program was set up with a static username and password (resepctively 'admin' and 'Admin123'), and we were never prompted to change this. This means that if the system installer doesn't notice this is the case, the is a user with administrator priviledges available to anyone, which is pretty bad. 2) The system didn't work with the newest version of MySQL. The latest version it can run (newest 5.6 version) has several vulnerabilities, some very serious.[1] While some of these also are present in newer versions of MySQL, others are not, making the database slightly more vulnerable.

[1] CVEDetails. *Oracle MySQL Security Vulnerabilities.* `https://www.cvedetails.com/vulnerability-list/vendor_id-93/product_id-21801/Oracle-Mysql.html`. Accessed: 04. Oct. 2016. 2016.

Chapter 3

Tests of the software tools

After we set everything up, we ran a test run of each analysis tool, to make sure everything was functioning properly and to find any glaring issues, if any.

Dynamic analysis of the OpenMRS Web GUI The initial Quick Start scan of our test setup of the OpenMRS service only had the login front page to crawl. This yielded very few results considering this is just one page. It found some information about jQuery, ZAP also warned about the implementation of the cookie.

Audit Workbench report The first scan yielded a few messages of high concern. The OpenMRS core includes unit-test which gives quite a few erroneous warnings concerning security risks and more general bad coding practices.

There was an error concerning how the cookie was coded which may be interesting to look more into.

FindBugs code scan Running the scan seems easy and most of the work is obviously analysing the results. The reports might be a bit hard to navigate and there was some trouble generating a html report that potentially is more readable. Again the unit-test generated false positives.

ThreadSafe The initial scan gave a few possible errors, all of them concerning either the OpenMRS-API or the OpenMRS-WEB which might make sense as these expose surface to external modules. The scan also gave some false positives referring to unit-test.

3.1 Findings

3.1.1 Raw data

Fortify Audit Workbench found 60 critical vulnerabilities, as well as 115 issues considered highly vulnerable. Following is a list of the categories designated by Audit Workbench for these vulnerabilities, as well as the number of occurrences of each vulnerability and a short explanation.

The critical vulnerabilities were in the following categories:

- Command injections (4): The program tries to call an external command without validating where the command is from.
- Insecure SSL (3): Uses SSL certificates are based on default system Certificate Authorities, which may be compromised, and sometimes are.
- Hard coded passwords (3): Lets an attacker log in if he or she has access to the source code, which in this case means anyone.
- Path manipulation (2): The program accesses a file with a file path that can be chosen by the attacker.
- Privacy violation (49): The program mishandles confidential information in some way or another, for example by writing it to disk.

The vulnerabilities rated high are as follows:

- Header manipulation with cookies (1): The program doesn't validate data in an HTTP request, so cookies can be tampered or spoofed.
- Hard coded encryption key (3): Same issue as with hard coded passwords.
- Log forging (12): The program doesn't validate inputs that get written to the log, meaning malicious data could be entered.
- Empty passwords (6): The program assigns empty passwords, making it much easier to break into the application.
- Hard coded passwords (25): Same issue as with the critical hard coded passwords.
- Passwords in Configuration File (36): Passwords are stored in plain-text in a configuration file, meaning an attacker can get the password by getting the file.
- Privacy violation (10): Same issue as in the critical privacy violations.

- Privacy violation from heap inspection (3): The program stores confidential information in strings, which are immutable and thus stored in memory until the JVM garbage collector is run (which may not occur for a while).
- Race Conditions from Singleton member field (19): The program uses a singleton class which takes concurrent connections, meaning data can be shared between users.

FindBugs The first scan detected 1058 potential bugs. Some of them are from test classes and will not be mentioned here. The report generated from the FindBugs GUI was very uninformative and hard to navigate. We found a plugin for IntelliJ IDEA that used Findbugs as the back-end and produced a more informative report where it was easy to exclude the testbase as IntelliJ had generated a build of the codebase giving a sense of which code was where. In this new report we had a total of 296 warnings and a separation between what FindBugs was very certain was a bug and what maybe could be a bug. In the following text we have made a presentation of the bugs FindBugs is most certain is a potential problem.

Correctness

- Masked Field:

  Class defines field that masks field in superclass org.openmrs.web.filter.StartupFilter: This error is mostly harmless. It may have potential for generating errors in later updates to the code. The field is marked as protected in the superclass so it is excessive in the child and probably should be removed.

- Masked Field:

  Field UpdateFilter.log masks field in superclass org.openmrs.web.filter.StartupFilter: Same as above.

Bad practice

- Dropped or ignored exception:

  This method might ignore an exception. In general, exceptions should be handled or reported in some way, or they should be thrown out of the method.

  This error occurred in the following classes:

  - ServiceContext.setModuleService
  - WorkflowCollectionEditor.setAsText
  - OpenmrsUtil.parse
  - BooleanConceptChangeSet.getInt
  - BooleanConceptChangeSet.createConcept

- Initialization circularity.

  Class makes active use of subclass during initialization and might result in null value.

  Class:

  - Operator

Internationalization

- Reliance on default encoding. Found code that does a bit to string conversion while assuming the default platform encoding is suitable. This could make the code behave differently on different platforms.

  Classes:

  - HL7ServiceImpl
  - ModuleFileParser
  - ModuleUtil
  - TextHandler
  - DatabaseUpdater
  - HTTPClient
  - OpenmrsUtil
  - CreateCodedOrderFrequencyForDrugOrderFrequencyChangeset
  - MigrateConceptReferenceTermChangeSet
  - SourceMySqldiffFile
  - Listener
  - StartupFilter
  - TestInstallUtil

Malicious code vulnerability

- Mutable static field. Public static field isn't final and could be change by malicious code or by accident.

  Classes:

  - SchedulerConstants
  - OpenmrsConstants
  - Security

- Field is mutable array. Same as above and field is referring to an array.

  Class:

  - OpenmrsConstants

Multithreaded code vulnerability

- Unintended contention or possible deadlock due to locking on shared objects: Synchronization on Boolean. There is a risk of syncing on a object belonging to another process since there is only two Boolean objects. Resulting in a deadlock

  Class:

  - HL7InQueueProcessor

- Unsynchronized Lazy Initialization: Incorrect lazy initialization and update of static field. Static field is not completed and could be accessed by another process while it is still initializing which would lead to a multithreading bug.

  Class:

  - OpenmrsClassLoader

Performance

- Questionable Boxing of primitive value. Boxing a primitive to compare. It's more efficient to use static compare method on primitives.

  Class:

  - ModuleFactory

- Boxing/unboxing to parse a primitive. A sting is boxed just to be unboxed for parsing. More efficient to use static parse*** method.

  Class:

  - BaseHyphenatedIdentifierValidator

Security

- Potential SQL Problem. Nonconstant string passed to execute or addBatch method on an SQL statement The method invokes the execute or addBatch method on an SQL statement with a String that seems to be dynamically generated. Consider using a prepared statement instead. It is more efficient and less vulnerable to SQL injection attacks.

  Classes:

- ConceptValidatorChangeSet
- MigrateAllergiesChangeSet

Dodgy code

- Dead local store. This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used. Note that Sun's javac compiler often generates dead stores for final local variables. Because FindBugs is a bytecode-based tool, there is no easy way to eliminate these false positives.

  Class:

  - PatientServiceImpl

OWASP ZAP A passive run of ZAP returns the following four possible vulnerabilities:

- X-Frame-Options header is not included in the HTTP response to protect against 'Click-Jacking' attacks: Considered a medium vulnerability, this vulnerability can let unknown sites be loaded inside frames or iframes.
- A cookie has been set without the HttpOnly flag: A low vulnerability that makes it possible to access the cookie using Javascript.
- Web browser XSS protection not enabled: A low vulnerability where the HTTP header X-XSS-Protection isn't set, so that the browser doesn't use it's own cross site scripting protection.
- X-Content-Type-Options Header missing: Some older browsers try to automatically figure out the content type if X-Content-Type-Options is not set to 'nosniff', making it possible to do such operations as cross-site scripting.[1]

When re-running ZAP and giving it access to an administrator user, as well as manually performing the scans, another couple of vulnerabilities show up. The new list contains the following, in addition to the previously mentioned:

- SQL Injection: Considered a high vulnerability, it can grant access to an attacker through manipulation of SQL queries in input forms.
- Format String error: Considered a medium vulnerability, it happens when an input string is evaluated as a command.

[1]Microsoft. *MIME Type detection in Windows Internet Explorer*. `https://msdn.microsoft.com/en-us/library/ms775147.aspx`. Accessed: 21. Oct. 2016. 2016.

- Incomplete or No cache-control and Pragma HTTP Header Set: Considered a low vulnerability, can let both the browser and proxies cache content
- Cross-domain JavaScript source file inclusion: Considered a low vulnerability. Is reported because ZAP has found a site using a script from another party.

ThreadSafe Excluding false positives from tests the scan gave 12 major and 1 minor warnings. No Critical or "Blocker" warnings were generated. The major warnings are as follows:

- Inconsistent synchronization of access to Field/a collection.

  StartupFilter, UpdateFilter.
- Mixed synchronization of access to a field.

  OpenmrsClassLoader, OrderSetServiceImpl, UpdateFilter.
- No lock held while iterating on a synchronized collection view.

  UpdateFilter.
- Non atomic use of Get/Check/Put.

  WebModuleUtil.
- Synchronizing on reusable objects.

  ModuleFactory, WebModuleUtil.
- Unsafe iteration over synchronized collection.

  HL7InQueueProcessor.
- Unsynchronized access to field from asynchronously invoked method.

  WebModuleUtil.

3.1.2 Interpretations of the raw data

Tests Some of the critical errors found by both Fortify and FindBugs were in test classes, and are therefore not going to run while the application is deployed. These bugs and vulnerabilities are therefore disregarded.

External files The vulnerabilities concerning passwords in clear text found by Fortify are mostly false positives, as they are found in localization files, and don't contain real passwords (instead containing the translation of the noun password). We will therefore disregard any vulnerability found in localization files. However, some files containing clear text passwords are used for setting up the server, and are thus relevant.

Privacy violations Many vulnerabilities found by Fortify were in the category Privacy Violations. Considering the type of application OpenMRS is (it manages patient journals), these are more important than for other applications, as patient journals contain lots of personal and sensitive information such as (possibly) social security numbers and medications used. Many of these files are about the the application logging login information and search terms, and the security of the logger is therefore of particular interest.

The other privacy violation needs the attacker to access the JVM memory, and therefore requires the attacker to have root access. Since the program runs on a server, if the attacker has root access you've probably lost your database and similar serious problems, and thus problems requiring memory access aren't as important. This might be more of a problem if the server is external (e.g. hosted on Amazon AWS or CloudFlare), but then, it is up to Amazon or CloudFlare

to secure their servers.

Misleading results Especially in the ZAP scan, multiple results, such as some (but not all) of the vulnerabilities enabling ClickJacking attacks, are found in websites related to the browser used and in the ZAP addon for Firefox. We will therefore disregard all reported ZAP vulnerabilities that contain no mention of OpenMRS.

Race conditions Fortify found indications of problems relating to the use of singleton. This can cause resources to be shared between users. This is both a privacy problem and a possible corruption of the integrity of data fields.[2]

ThreadSafe also pointed to some inconsistencies in handling fields and collocations of data. Most interesting is the two classes ModuleFactory and WebModuleUtil which seems to be operating somewhere between the OpenMRS core and possible external modules.

Dynamic analysis The run of OWASP ZAP didn't find any critical or highly vulnerable issues,

indicating that the front end is relatively well secured. The ZAP issues were related to HTTP headers (for example no SQL injection issues and direct XSS issues). Since the critical issues were found by static analysis, we will focus more on the server errors.

3.1.3 Vulnerable external libraries/software

In order to find vulnerabilities in libraries and software OpenMRS is based on we looked at exploit-db.com, which is a database looking up vulnerabilities in the CVE and OSVDB databases.

[2]OWASP. `https://www.owasp.org/index.php/OWASP_Periodic_Table_of_Vulnerabilities_-_Race_Conditions`. Accessed 22. November 2016. 2013.

We looked for exploits in Tomcat, JRE, Java, and MySQL. We found the following relevant exploits:

- Tomcat: There are issues with the newest version of Tomcat (Tomcat 8) for OpenMRS, and we therefore looked for exploits in Tomcat 7. There are especially two exploits worth noting:
  - EDB-ID 35011: Apache Tomcat 7.0.4 - 'sort' and 'orderBy' Parameters Cross-Site Scripting: Since Tomcat doesn't sanitize user inputs properly, cross-site scripting can be performed.
  - EDB-ID 40450: Apache Tomcat 8/7/6 - Privilege escalation: The default Tomcat distributions available on Aptitude (Debian and Debian based distro package manager) as well as RPM (RedHat and RedHat derivate distro packagr manager) have a systemd config file that is insecure, run on boot, and can give root access to an attacker. The payload for this file can be delivered through a Tomcat shell in java web apps, or locally. This is a new exploit (found 10/10/2016) and thus unlikely to be patched on servers running OpenMRS.
- MySQL: EDB-ID 40678: MySQL/MariaDB/PerconaDB System user Privilege escalation / Race condition: Lets a local user access and run database operations above their own privilege level, and combined with other exploits, can give server root access to a user.
- Java/JRE: OpenMRS runs on JRE 1.7, for which we couldn't find any relevant exploits.

[3]

[3]exploit db.net. `http://www.exploit-db.net`. Accessed 16. November 2016. 2016.

Chapter 4

Analysis, evaluation and recommendations

4.1 Evaluation of tool reports

Audit Workbench Audit Workbench presented us with 60 critical errors, of which nearly all were found in tests. However, the vulnerabilities related to SSL certificates are not related to tests, and would let an attacker spoof a certificate and insert itself between the server and the endpoint. OpenMRS would trust the spoofed certificate and therefore let the user recieve unauthorized data. This is critical for instances of OpenMRS connected to the internet.

There are also several issues in the "high" category: For example hard coded encryption keys, meaning anyone with access to the source code (which in an open source project is anyone) has access to the encryption key. Since the project uses AES encryption by default, which uses the same key for encryption and decryption, it means anyone has access to the decryption key. Another issue is that some default passwords are stored in cleartext, and while the application asks for the system administator to change default passwords, the task is delegated to the user, which may or may not comply. It would have been safer not to store any passwords in cleartext.

FindBugs Most of the FindBugs errors were in the category Malicious Code Vulnerablity warnings, which, along with Security Warnings, is the most relevant category for our purposes. The warnings given a high priority by FindBugs are warnings about non-final properties that should be final. According to the description from FindBugs, these fields could be changed by malicious code or by accident by programmers working on OpenMRS or an OpenMRS module. Among the properties not set as final are properties setting the data directory OpenMRS uses, as well as a regular expression and other password requirement settings. If these get changed by accident or on purpose, it could let an attacker use either point the software to a location on the

computer under his control, or by setting a custom regex, whitelist characters that would let the attacker perform SQL injection attacks.

FindBugs also presents many warnings about OpenMRS returning references to mutable objects, something that can let an attacker understand how OpenMRS represents internal state and data. Again, this is mostly relevant if the attacker writes a module for OpenMRS that gets installed and for example leaks information about the running state of the application to the attacker. FindBugs recommends returning a copy of data instead of mutating data in most cases.

OWASP ZAP ZAP reported an SQL injection which was found in the Forgotten Password page of OpenMRS. As this page is available to all persons, both users and non-users, it is a quite significant vulnerability which can grant an attacker access to the user database. ZAP also reported multiple examples of ClickJacking vulnerabilities. Some of them are not in OpenMRS itself, but rather in a plugin for Firefox used by ZAP, and can be disregarded. Others are, however, in OpenMRS. Depending on the functionality in OpenMRS added by modules ClickJacking can be an issue, as it requires overlaying some sort of CSS and/or Javascript to trick users into clicking on something they did not intend to click. Some kind of XSS attack would therefore be needed. If for example a module is added to OpenMRS that recieves emails form patients and lets a doctor answer them, an attacker could perform a ClickJacking attack by overlaying an iFrame in the sent message and sending it to the doctor. ZAP recommends setting X-Frame-Options to for example DENY. OpenMRS has not set this header by default.

ThreadSafe As Fortify flagged a possible problem with the implementation of the singleton in openMRS with the possibility of information being shared between users we felt the need to dig deeper in the possibilities of problems in the concurrency. We introduced ThreadSafe as a dedicated tool to statically analyze the codebase, as concurrency is a complicated issue we thought a tool that was dedicated to searching for problems with concurrency would give results that could be trusted more.

ThreadSafe pointed to inconsistencies in accessing fields and lists in several classes. There seems to be inconsistent locking of the libcachefolder in the openmrsClassLoader. Most interesting seems to be the number of inconsistencies in locking in the ModuleFactory, WebModuleUtil and in some of the filter classes. There are many classes in OpenMRS that uses the java.util.concurrent library to handle locking of resources, tho none of them seem to be a part of the OpenMRS web library. As modules are a key element in the operation of OpenMRS is seems critical that a collection of classes that relate to each other, as the ModuleFactory most likely is related to the WebModuleUtil and WebModuleUtil uses filters and initiates singletons, all seems to contain the same possible error types and these types are crucial to runtime operation. All of which significantly increases the possibility of a problem materializing creating problems during operation.

4.2 Evaluation of external libraries

MySQL Being able to access and edit the database is very dangerous in a system that contains such sensitive data as patient journals. As noted, the EDB-ID 40678 vulnerability in MySQL could be used to gain root access to the database. It requires local access, but since OpenMRS can be used for example in a hospital with a lot of local users of different privileges (for example Medical Student, Laboratory Assistants and of course Doctors), the vulnerability is very relevant to OpenMRS.

Tomcat Since a lot of servers running OpenMRS are Linux servers running e.g. Ubuntu (the official installation instructions list Windows and Ubuntu installation steps), EDB-ID 40450 is a dangerous exploit. It can be exploited through a web shell delivered through for example cross-site scripting or SQL injection, and can give root access to the server, giving the attacker full control.

4.3 Recommendations

4.3.1 Recommendations for developers

OWASP Top 10 Of the errors found by Fortify, most were in the current (2013) OWASP Top 10 category of Security Misconfiguration, which includes default accounts that are enabled and unmodified. As previously stated, there were examples of default administrator users being asked to be removed by the system administrator. The safer approach would be to force the sysadmin to create a new administrator-level user and then automatically delete the default one. The same goes for database access credentials, and maybe also encryption keys, especially if the default enctryption method is AES. These vulnerabilities are, according to OWASP, easy to exploit, but also quite easy to find, and would therefore be a good target for something the developers could improve on, even just by providing documentation to system administrators of how to perform scans checking whether default values are in use.

OpenMRS, using default SSL certificates for the system, could also be considered as part of the OWASP category Broken Authentication and Session Management, which has become the second most common vulnerability on the list. According to OWASP, these types of vulnerabilities can have severe consequences, letting an attacker gain control over any user account, and possibly even all user accounts. These errors are more difficult to exploit than the Security Misconfiguration ones, but also more difficult to find. The OWASP recommendations for this category of vulnerabilities is more diffuse, especially for the specific issue of SSL certificates. The entry for Broken Authentication and Session management also links to the category Sensitive Data Exposure.

Sensitive Data Exposure considers cases of data being transmitted in a way which can be stolen. Fortify lists just over 60 privacy violations that are either critical or high on the vulnerability spectrum, and some of these are found in such classes as HttpClient, which handles HTTP POST data, and which doesn't encrypt sent data. There are also multiple places the application logs things such as the username performing an operation. While this can help find the identity of an attacker, it also lets an attacker find information about the registered users, easing for example bruteforce attacks. According to OWASP, Sensitive Data Exposure should be mitigated by encrypting all sensitive data that's sent, as well as not storing sensitive data unnecessarily

ZAP found one SQL Injection vulnerability. Injection vulnerabilities are the most common category on OWASP, and have been so for a long time. SQL injection attacks are easy to perform and can have a severe impact. In this case, the vulnerability was found in the Forgotten Password page, which is accessible to non-logged in users, and is therefore especially dangerous. As OWASP recommends for mitigating injection attacks, there should be server side evaluation of form input and all special characters should be escaped on the server.[1]

Race conditions Since race conditions are becoming an increasingly more commmon cause of errors and can cause such errors as returning critical data to non-authorized users, these errors are a potentially important area for the developers to improve. It seems that the java library for concurrency is the one used in the codebase. So as the ThreadSafe report stated there seems to be a collection of potential bugs aggregating around handling of the modules and there might also according to the Fortify rapport be some misconception around how the singletons operate and when they are used. Then a possible change to take into consideration is to implement the java concurrency utilities library in the OpenMRS-Web library and especially around the handling of modules and filters.

4.3.2 Recommendations for users and implementers

Recommended software The install instructions for OpenMRS links the user to Tomcat 6, and with a generic statement about how "there are issues with Tomcat 8.". In light of the Tomcat exploit found on Exploit-DB, the user should not be using a version of Tomcat from the package manager of the operating system if running a Linux server. Downloading the newest version of Tomcat 7 from Apache's homepage is therefore the reccomended course of action.

User management OpenMRS has three unchangeable user groups, Anonymous, Authenticated and System Developer. Floss Manuals (which the official OpenMRS wiki refers to), there are some specific recommendations for the privileges these users should recieve. From Floss-Manuals: "Anonymous privileges should be kept very restricted, since patient information might

[1] OWASP. `https://www.owasp.org/index.php/Top_10_2013-Top_10`. Accessed 21. November 2016. 2013.

otherwise be compromised. Privileges granted to the Authenticated role are granted to anyone that logs into your system, no matter what other role(s) they might be assigned. Granting privileges to the Authenticated role is an easy way to grant privileges to all users of the system. The System Developer role is automatically granted full access to the system and should only be granted to system administrators.

Super users (system administrators) are automatically granted all privileges in the system; therefore, you must be very careful to protect your system administrator password."[2] We recommend that users of the application follow these guidelines, as not complying can lead to granting privileges to all users of the system, ergo more easily letting attackers gain information not usually available to them.

Also, as noted earlier, system administrators should take extra care to remove default users such as the user "admin" with the password "Admin123".

4.4 Summary

The static analysis was done with the tools Fortify, FindBugs and ThreadSafe. All of the tools were easy to set up and the initial scans were easy to initiate, except Fortify which required a bit of tinkering to allocate enough memory for it. All of the tools have more advanced features and the possibility for tailoring the settings more appropriate the needs of the project, both of which we did not have the time or experience to do.

A note on FindBugs: During our project the maintainer of FindBugs discontinued it. This did not affect our project, but an alternative for future projects might be SonarQube, which is another static analysis tool that seems more user-friendly and with more complete results(bigger rule collection).

The dynamic analysis tool we used was OWASP Zap. This program has a steeper learning curve and the results were a lot harder to analyze. For a more effective use than what we achieved there is a need for a better understanding of functionalities the program offers at the same time to have a good understanding of the results there is a certain amount of understanding of the program need.

Even with our short time using the tools the effort did return interesting results in all areas of the codebase giving rapports of potential errors of very different nature, for example concurrency vulnerabilities and weakness in design of users and roles. Especially the concurrency vulnerabilites seem prevalent. Since OpenMRS is a Java application and concurrency is not a first-class citizen in Java it is essential to have a good tool for finding errors in the implementation of concurrent classes, as the bugs found by ThreadSafe confirms.

[2]FlossManuals. *User Management and Access Control*. `https://write.flossmanuals.net/openmrs/user-management-and-access-control/`. Accessed: 14. November 2016.

The potential weaknesses we found in the design of roles and users could not be discovered with static analysis alone. We recognize that automatic tools for analysis is not enough, so to discover faults in the design manual analysis and learning how the code is implemented is necessary.

To finish off, we would like to state that since this is our first foray into security testing, the results should be taken with a grain of salt, and might not reflect the real state of security in OpenMRS. However, we feel that we found relevant issues in many areas, and consider the analysis a success.