INF226 Obligatory assignment

Håvar Eggereide and Syver Storm-Furru

2016-10-02

Contents

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?

Is 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.

2 Software involved

2.1 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.

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

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.

2.2 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).

2.3 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.

2.4 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]

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

[4] University of Maryland. *FindBugs - Find Bugs in Java Programs.* 2015.

[5] OWASP. *OWASP ZAP 2.4 Getting Started Guide.* 2016.

We ran ZAP version 2.5.0 for our tests.

3 Preliminary results

3.1 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.[6] While some of these also are present in newer versions of MySQL, others are not, making the database slightly more vulnerable.

3.2 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.

[6] 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.

4 Findings

4.1 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.

Configuration Files

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 to IntelliJ IDEA that used Findbugs as the back-end and produced a more informative rapport where it was easy to exclude the testbase as IntelliJ had generated a build of the codebase giving at a sense of which code was where. In this new rapport 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 isnt 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. Its 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 'ClickJacking' 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.[7]

4.2 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. This especially applies to the Fortify results in the category Command Injection: The application runs commands from external sources without validating them beforehand.

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.

[7] **XFrameOptions**.

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.

Race conditions These errors arise from Servlet being a singleton. Since the application changes member fields of this singleton, multiple request to get information can return

wrong results (if the member field is changed during requests).

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.