ICS491 Online Timesheet Server Software: Secure Development Lifecycle

Authors: Marco de Lannoy Kobayashi, Collon Saphore

Date: July 16, 2017

Abstract: We plan on creating a timesheet cloud service that employers may utilize to outsource their timesheet operations and storage. This report is the first part of a series that will establish the Secure Development Lifecycle process that will be implemented to ensure that our final product maintains confidentiality, integrity, and availability at necessary high-quality levels.

Github Repo: https://github.com/marcodlk/ics491_timesheet

1 Introduction

We are developing a web-application in Java with queries to a DB2 database using mySQL. The application will be an online timesheet program where administrators may register users as employees, and modify or delete current employee user accounts. Registered users will be able to log hours. Users will have a username, password, and possibly additional metadata. Information will be stored in a DB. The program will allow a registered user to clock in/out. Employee information can be requested (Ex. Hours logged today, this week, on average, current salary, etc). We will be following the Microsoft SDL process as our framework for secure code development.

2.1 Establish Security Requirements (by Cameron Simao)

For our program, there are many security and privacy requirements. There are three levels of priority in our security. Priority 1 code includes that which requires the highest level of security. These things include:

- Code that deals with the database
- Code that involves the network
- Code that is running while running with admin privileges

Priority 2 and 3 code includes the rest of the code that is respectively less important or susceptible to attack. This includes items such as setup code for the program and optional features that don't need to be run.

There are many bug causes that we need to worry about such as weak authentication or not enough error messages. To keep track of security flaws, we can run ideas by each other to figure out which area in the program is the most vulnerable. Also, user testing can allow us to

find other vulnerable areas. After gathering information from these methods, we can reference our priority list and deal with the problems based on their priority level.

2.2 Create Quality Gates / Bug Bars (by Cameron Simao)

Security Bug Bar	
Critical	The server is taken over by some unauthorized source ● Someone has a higher access level than they are supposed to ○ Remote Anonymous User ■ SQL injection ■ Using arbitrary code ■ Writing to the file system with arbitrary code ● Security Features ○ Bypassing any type of security feature such as a firewall
Important	● Information disclosure ○ Attacker can get information from anywhere in the system (database) such as credit card information or other personal information ● Spoofing ○ Attacker disguises as someone ● Tampering ○ Attacker changes files or database entries
Low	● Information disclosure ● Tampering ○ Changes in data that doesn't stay after restarting the program

Privacy Bug Bar	
Critical	● Lack of notice and consent ○ Personal information being transferred without complete consent from the user ● Lack of user controls ○ User should be able to stop any transfer of personal information ● Lack of data protection ○ Personal information should be protected with some form of authentication ● Lack of legal controls ○ Third parties shouldn't be able to get any information if they are not in contract. ● Lack of enterprise controls ● Insufficient privacy disclosure
Important	● Lack of notice and consent

	○ Same as in “Critical” section ● Data minimization ○ User data is given to third parties for a different reason than stated ● Lack of enterprise controls ● Insufficient privacy disclosure
Low	● Lack of notice and consent ○ Personal information is hidden and cannot be accessed by others ● Lack of enterprise controls

(Framework borrowed from [3] and [4])

2.3 Perform Security and Privacy Risk Assessments

Below we detail some security and privacy risk assessments that we made:

- Does not store personal information on user’s computer but rather a server that stores all information. Users interact through a java application.
- Since this is based on employee records and timesheets, there is no basis from which to target children.
- There’s no information this program receives from monitoring. There is a sign in and sign out, with some functions to retrieve employee information for the employee and the employer.
- This program should not be installing or modifying any files on any computer.
- Does not transfer any anonymous data.

Despite what looks to be a P3 Scenario, we determined that the sensitive information on the database requires P1 Scenario considerations as well, such as how data is stored or transferred; what, if any, auxiliary software is installed or required; how users will access public disclosure; how organizations and users can control our feature; and how unauthorized access will be prevented.

We store username and passwords associated with logins. These reflect to personal information about the employee. From user hours, to employee number, and more sensitive information like government document numbers and social security numbers.

In order to use the application, one is only required to have an updated version of Java.

Users will access our public disclosure through a Java application which will interact with a database through queries. Administrator users (i.e. employers) will have more extensive access of the database and actually commit changes.

Each database belongs to one company for a one to one relationship. These organizations will have almost complete access to features at the top level, and restricted in access the further down the ranks you are. However, organizations will not be able to design their own database. We will have preset information the organization may store with our application. Any other private information is upon the organization's responsibility to maintain secure through provided security protocols.

Users such as employees will be able to login and query the database for information. The only commit they will be able to make is a clock in and out.

Usernames and passwords will match to exactly one row of PII in the database. Users of certain levels (Ex. head of the organization) will have access to all database information. However, basic users will not be allowed functionality that queries outside of their own row.

3.1 Establish Design Requirements

These requirements are focused on attack surface reduction (ASR). Thinking critically about limitation and verification is important to a safe program. To decide which implementations provide a high attack surface vs a low attack surface can be difficult when

balancing the best product efficiency possible. Sometimes high risk implementations are also just the more sensible way to go. However, in agreeance with Microsoft's Basics of Secure Design, "ASR is just as important as trying to get the code right."

There should only be entry from Network I/O, in the form of queries and commits. No file's will be allowed to be sent, modified (other than the database itself), downloaded, or removed as a requirement of this application. All users should be authenticated, as they must log into an account with a username and password. There will be administrator and user accounts with varying levels of access to the application.

More specific design requirements such as password specifications must be in place. For example, no duplicate usernames are allowed to exist. Passwords must also require Uppercase, lowercase, number or symbol, and be at least 8 characters in length. Further authorization will be required for administrator accounts to provide the identity and legitimate use of the account when requesting more sensitive information.

3.2 Perform Attack Surface Analysis / Reduction

The final product will contain two distinct privilege levels: employee and administrator. Employee users will be limited to checking in and out and requesting information pertaining to the employee's own records and timesheet data. Administrator users will have the power to add or modify, including deleting, records as well as requesting information about any appropriate set of employees. There may be varying levels of administrator privilege, as determined by the "customer in chief". This initial report analyzes the attack surface consider a singular (default) administrator privilege level.

Let's begin by taking a look at the attack surface pertaining to the employee user. These simple user accounts do not require any custom user input due to functionality being limited to

checking in and out and making preset requests. This allows for a design that neglects the user the ability to enter custom input, thereby greatly reducing the attack surface here.

The real point of interest in this attack surface analysis is that pertaining to the administrator. Here multiple factors must be considered; not only potential software flaws but the human factor as well. With the ability to modify and delete records, administrators have the ability to wreak havoc on a database if the proper policies and programmed measures are not established. Penetration testing should prioritize considering possible attack vectors that would allow a malicious actor to infiltrate the system with administrator privileges.

Below is a list of what we deemed to be points of highest interest to attackers and highest focus for security orientation given the specifications of our application:

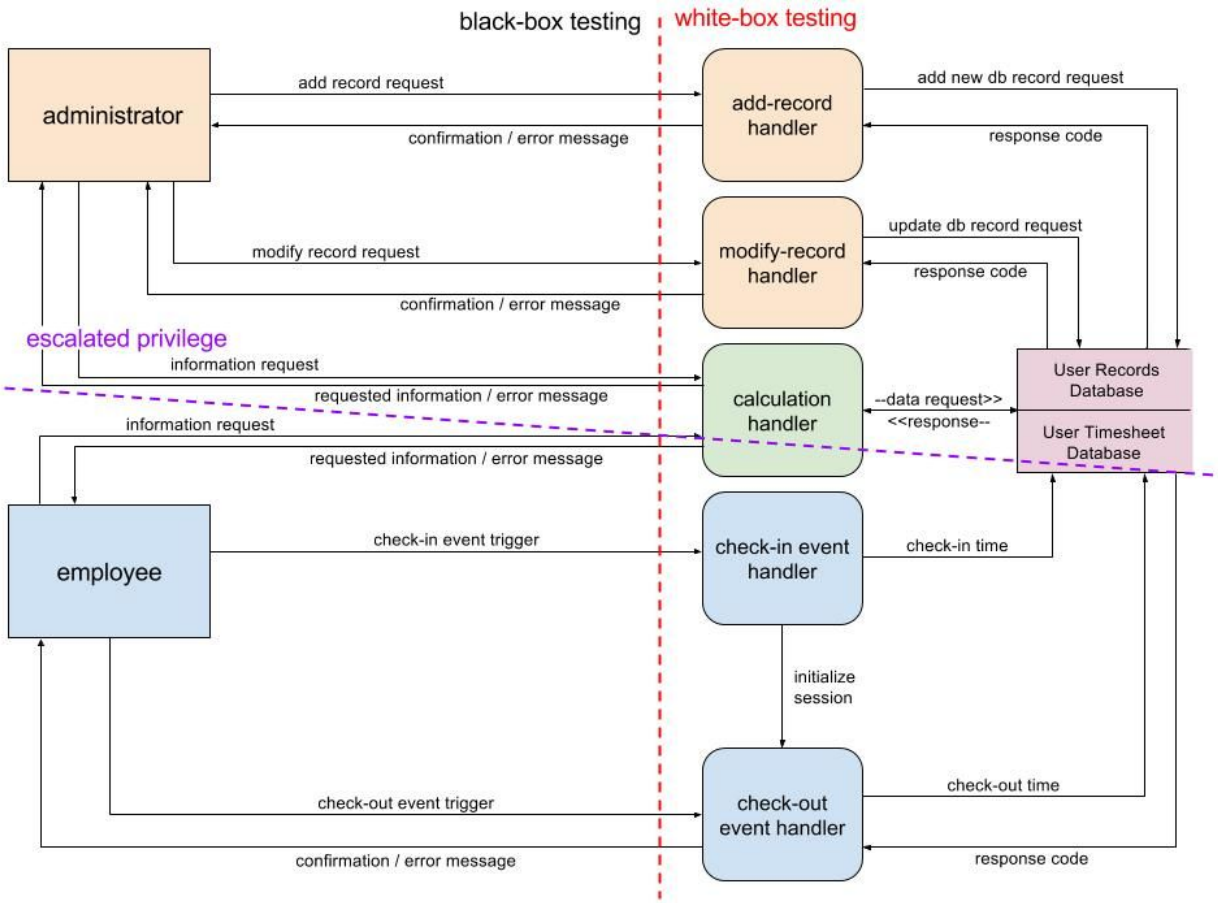
- Authentication entry points
- Authorization bypass points
- Administrator interface
    - record database modification input forms
    - searches and inquiries
- User records database
- Error response system
- Data flow and processing in response to information request
- Calculation handler where the admin and employee privileges theoretically “touch”

3.3 Use Threat Modeling

Analyzing the DFD (Data Flow Diagram) below we can identify that the calculation handler is an area of interest due to its existence at the boundary between privilege levels (purple dotted line). Employee users should only be able to make information requests pertaining to their own records in the database, but the same handler is also capable of

retrieving information from other records given proper authorization. The handler is at risk of being exploited to escalate employee user privileges up to administrator privileges. Proper secure coding strategies and penetration testing must be focused on this node.

For penetration testing, it is also valuable to note the distinction between black-box testing and white-box testing components as indicated in the DFD below (red dotted-line). The components to the right side of the line reveal the application's internal design and will be vital in white-box testing for software flaws. Components to the left provide only the view of registered users and are useful to give black-box testers the respective vantage point in search for exploitable entry points. Authentication and authorization mechanisms are not integrated in the DFD below but are considered high-value targets for malicious actors.



4 Implementation

To ensure that our implementation follows secure coding practices, we have compiled the following lists of approved tools and unsafe/deprecated functions. Please note that the unsafe functions table discusses functions that may be required to use due to vital functionality but we do list coding practices that aim to circumvent potential vulnerabilities.

We also discuss the static analysis tool we are integrating into our code development, its current impact, our experience with it so far, and how we plan to progress.

4.1 Approved Tools

Java

Compiler / Tool	Minimum Recommended Version	Comments
javac	1.8.0_25	
Java™ SE Development Kit 8	Update 141 (JDK 8u141)	Blocks SHA-1 TLS certificates by default. Valuable once the server application goes online.

Database

DB2 LUW	11.1	
MySQL	5.7.17	This update had significant security fixes.

Server

Apache Tomcat	8.5.16	

Static Analysis

Google Checkstyle	8.0	

4.2 Unsafe/Deprecated Functions

Java (sql,jdbc,oracle) - deprecated

- oracle.sql.StructDescriptor.createDescriptor - It is recommended to use connection.createStruct instead.
- oracle.jdbc.OracleConnection.clearAllApplicationContext - It is not recommended to use this alongside get/setClientInfo APIs.
- oracle.jdbc.OracleStatement.defineColumnType(int, int, int, short) - use defineColumnType(int, int, int)
- oracle.jdbc.OracleStatement.defineColumnTypeBytes(int, int, int) - use defineColumnType(int, int, int)
- oracle.jdbc.OracleStatement.defineColumnTypeChar(int, int, int) - use defineColumnType(int, int, int)
- oracle.sql.DatumWithConnection.getConnection - use getJavaSqlConnection(), getInternalConnection(), or getOracleConnection() instead.
- oracle.jdbc.OracleConnection.getJavaObject
- oracle.jdbc.OracleConnection.pingDatabase
- oracle.sql.TIMESTAMPLTZ.(toString|toBytes|toDate|toTime|toTimestamp)

Java - unsafe / require attention

Category	Function	Comments
ClassLoader	java.lang.ClassLoader.defineClass java.net.URLClassLoader	- Allows arbitrary java code execution + Use java.security.SecureClassLoader + Validate URLs used with URLClassLoader + Limit exposure of ClassLoader instances
File Access	java.io.File (constructor) java.io.File.delete java.io.File.renameTo java.io.File.listFiles java.io.File.list	- Allows file renaming and deletion - Allows access to directory listing + Enable java.security.manager
File Stream Reader/Writer	java.io.FileInputStream java.io.FileOutputStream java.io.FileReader java.io.FileWriter java.io.RandomAccessFile	- Allows reading and writing access to files + Enable java.security.manager
System Property	System.setProperty System.getProperties System.getProperty	- Allows access to system properties, which may contain sensitive information + Enable java.security.manager
OS Executables	Runtime.exec ProcessBuilder	- Allows arbitrary code execution + Sanitize input to ensure that shell cannot be invoked, which then allows for RCE
Load Native Libraries	System.load	- Allows arbitrary code execution

	System.loadLibrary	through corrupt loaded libraries + Protect library files to be loaded

4.3 Static Analysis

For static analysis, we are using the Checkstyle for Java tool, following the Google Checkstyle standard. We did not enlist the checkstyle tool from the beginning of development, which was a mistake. We prioritized developing an initial bit of code that manifested our design map in a very primitive yet foundational way. Although we succeeded in our goal, upon employing the checkstyle in a procrastinated fashion, we became overwhelmed with unmet guidelines and necessary revisions. At the time of writing, we are still working on aligning our code with the checkstyle guidelines and hope to move forward in a more checkstyle-integrated fashion. Interestingly most of the errors are small semantic issues, such as indentation with the tab character rather than spaces, or an expected newline to obey their strict code structure. Some of these suggestions are not security oriented but they are nevertheless important to produce consistent, readable, and legacy-facilitating code.

5 Verification

Considering we had to make some revisions that entailed backlogging web server integration to focus on other main functionalities of the program, a lot security issues were backlogged with that as well. Removing the program from the Internet immediately protects it from countless attack vectors we have been discussing, such as XSS, XSFR, cookie manipulation, and general Javascript exploitation techniques. However, we also had to decide to backlog the DB2 database integration due to some misunderstanding that culminated in an inability to move forward in this regard given the time and personnel constraints. Assuming the

project will continue after this initial summer deadline, we plan on integrating these vital components into the program before any final product release.

Unfortunately, for the moment, this means that our stripped down program will be using a local “database” in the form of CSV files, which comply with DB2 requirements. This database can be found in the “resources” folder adjacent to “src”. Clearly this is a tremendous security risk were this to be the final design (it is NOT). Nevertheless, for the purpose of the report at this time, the code will be considered in the state it is in, so keep that in mind as while reading the following three subsections, particularly Fuzz Testing and Attack Surface Review.

5.1 Dynamic Analysis

We were not able to integrate an official dynamic analysis tool for our code by the time of this report. We hope to employ the JUnit suite for unit testing but in the meantime we have stuck to the old fashioned strategy of performing the dynamic input ourselves.

5.2 Fuzz Testing

In the state the code is in at the moment, as a fuzz tester, the first thing I will notice is the juicy “database” directory located in “resources”. I am able to *cat* all the .csv files and see all the sensitive information inside the program’s database! Looking at users.csv, I notice there are two number columns; I can “guess” one of them is the user ID since it is unique to each user, and the other is possibly a privilege level identifier since there appears to only be two variations, 0 or 1. Another interesting column is one that appears to be some sort of hash. Dumping the exposed hashes to a file, I can run a dictionary attack to try to crack the password. Better yet! I can simply insert my own hashes to replace the ones there so that I can enter the program as any user. In fact, I can modify the database as I please and have full control of privilege elevation in the program. By creating my own row with a privilege level 0, I am able to run the

program and apply the appropriate credentials with admin access. This just goes to show the perils of having your database so exposed to any user.

Other than exploiting that very obvious vulnerability, the other two attempts we performed were unsuccessful due to it being a Java program, which means it is as secure as the JVM it runs on. One attempt was to cause a buffer overflow in the login prompt. we inserted a 1000 byte long string and were met with the same error message as with any other unacceptable entry. We tried this for both the username and password entries; both unsuccessful. Leaving the black box mindset for a second, it may have helped that we implemented the code with a maximum allowed username and password length.

The last attack we performed as fuzz testers was an attempt at an integer overflow at the main menu. We inserted numbers that surpassed the maximum values of signed and unsigned int and long but were unsuccessful; seeing the “unrecognized command” message again and again. This is due to the fact that the command is read and interpreted as a string, even though it expects the user to input digits. This may have been of concern and an easy target for buffer overflows in C/C++ but since we are using Java it is not a problem.

5.3 Attack Surface Review

As described in the introduction to this Verification section, we made some significant changes in our design goals for the short time allotted until the deadline. Deciding that we do not have the time and expertise required to properly establish the database on a web server by the deadline, means that our database will remain a small clutter of csv files local to the machine running the program. In a way, at this state, the security responsibilities are exported to the administrator of the program, who will have to ensure the proper restrictions are in place so that no employee users may go and edit the database at their will.

6 Incident Response Plan

6.1 Purpose

Our Privacy Escalation Team (PET) is composed of four main positions. The purpose of the PET is to identify the cause, discuss solutions and improvements to policies, manage internal and external communications. and resolve all concerns during an incident.

6.2 Responsibilities

Our current PET team is however made up of two members. We allocate the responsibilities of public relations and security engineering equally amongst the two members. Marco will be the head of the Team as the Escalation Manager. Collon will work as the Legal representative to field any questions from customers, and ensure proper use and distribution of product.

We each shall share the responsibilities of public relations representative and security engineer. As escalation manager, Marco will be in charge of delegating these duties when needed. With proper communication and planning, we can each assume these roles together to form a proper PET.

6.2.1 Duties of the Escalation Manager

Oversees the PET, deciding when it would be beneficial and necessary to include more members. Delegates responsibilities as needed when members of the team do not take initiation.

6.2.2 Duties of the Legal Representative

Ensures the proper and legal use of the product by clients. If any breach in security results in PII being stolen, this position will work with the public relations manager to do damage control for any clients.

6.2.3 Duties of the Public Relations Representative

Actively contacts customers for valuable feedback. Keeps an open line of communication for any concerns or complaints.

6.2.4 Duties of the Security Engineer

In charge of programming and testing. Actively tries to find vulnerabilities, run security tests, and improve design. Works with the Escalation manager during an extreme situation requiring a quick response.

6.3 Procedures

1. The PET team has just been alerted to a breach of security. If the Escalation manager was not the first to be notified, or a team member discovers the incident, the first step is to debrief him/her.
2. Each member of the team must then make time for a meeting to discuss the severity of the incident. Privacy Escalation Is an acceptable, but not limited to, response for the following incidents:
    a. Security breaches resulting in theft of PII
    b. Failure to meet security standards
    c. Privacy-related lawsuits
    d. Privacy-related inquiries from clients
    e. Requests for comments by media, government, or other agencies
3. Once determining severity and decision to move forward with a response, investigation must be conducted to identify the source and impact of the incident. A plan should be determined by the Escalation manager at this stage, before moving forward. Delegation of duties should occur.

4. Before an incident can be considered concluded, there must be a full review of policies and standards. A final outreach allowing an open line of communication to affected persons, both internal and external, must be conducted.
5. In concluding an incident, the Escalation manager must file proper documentation of the start to finish process. Each step of procedure should be met with satisfactory, with any discrepancies being noted in the incident file. This will help to improve on future incidents.

6.4 Contact the PET Team

Email: (contact-us@bogus.team.email)

Team individuals:

- Marco de Lannoy Kobayashi
    - Escalation Manager
    - Public Relations
    - Security Engineer
    - marcodlk@hawaii.edu
- Collon Saphore
    - Legal Rep
    - Public Relations
    - Security Engineer
    - csaphore@hawaii.edu

7 Final Security Review

7.1 Login Function

<u>Passed FSR w/exception</u>

- Improve hashing function
- Timeouts to prevent brute force

The username and password are stored in a separate database, contained with a user ID and a privilege level. Currently the password is stored as an MD5 hash and we compare at time of login. However there is pseudocode added to represent how we can improve the security. Using a global salt (that should be stored in a database as well) combined with using a per user salt would greatly increase the security of the log in.

Here is an example table to be used for a database.

table user (user_id auto increment integer primary key, username variable character [128], password binary 32, user_salt binary 8, privilegeLevel integer)

To explain:

This creates a table of users that has a user_id that is incremented from (0-X) where x is the number of users in the table. We can also use user_id to quickly find a user in a database..A username that is made up of characters, a password, a personal user salt, and the privilegeLevel to be referenced.

The user_salt is necessary for the use of password encryption. We are going to use an example of SHA256 and store a global salt to be to be used to with randomly generated 8bytes long. So we have as follows -> password encryption = SHA256(user_password + global_salt + user_salt).

Another implementation of security should be a time out function for users entering username and password. This would help to prevent brute force. We also prevent guessing password by not divulging any information about incorrect username and password combinations.

7.2 Punch Clock

Passed FSR w/exception

- Implement a way to restrict clock-in to assigned/expected IP locations

- Clock In/out database should not be local. Too easy for the user to find and manipulate.

Our clock in and out function is currently restricted to whatever computer has this program installed. If we were to allow for web application functionality, we would need to restrict users to only clocking in from approved sites. One way to do this would be to force a comparison of a user's session metadata against the referenced and accepted session values. So you could make sure that a user must log in from a selected IP address.

One important feature is our use of prepared statements. There are some sample mySQL statements that could be used for a web application that wanted to query and commit to a database. Using prepared statements sanitizes the input and help to guard from SQLinjection attacks.

7.3 Record Management

Passed FSR w/exception

- Admin is very powerful, this is dangerous

There doesn't seem to be any way to escalate the privilege from the token. This means that a normal employee should never have access to the same menu and functions as an admin. However an admin basically has access across the database inserting and removing rows. This means that anyone who gains access to this privilege can wreak havoc on the system.

7.4 Request Timesheet Info:

Passed FSR

- No current way to manipulate information

With the current functionality of requesting timesheet info, there is not much way to manipulate the information you receive from the database. Our use of a token is a good way to ensure that users don't get to decide which row they can get information from, without being given explicit permission.

7.5 Closing Thoughts

The premature and unsatisfactory state of the code at which we performed this final security review forced us to hedge our security expectations. This program is not ready for professional release. That being said, aside from some glaring issues such as the local, raw csv database, we did adhere to secure coding practices through development.

Drawing from our Threat Model developed at the beginning of the design phase, we incorporated both black box and white box testing respectively. Our black box testing was mostly successful (as in no successful exploitation) due to the fact that it is a Java application, meaning that it is as secure as whichever JVM it runs on (usually very). White box testing was very similar in results. Even knowing some of our program's potential security weaknesses - such as not strictly checking some user input for simple menu commands - due to Java's innate security checks, we were not able to manipulate the code to our advantage.

All in all, we reaped many benefits from programming in Java. After this report, proceeding in the direction we want to head, we will be forced to incorporate more strenuous security checks when we integrate true database and web server functionality.

8 References

1. "SDL PROCESS: REQUIREMENTS." *Microsoft*. N.p., n.d. Web. 16 July 2017. <https://www.microsoft.com/en-us/SDL/process/requirements.aspx>.
2. "Security Threat Models: An Agile Introduction." Agile Modeling. N.p., n.d. Web. 16 July 2017. <http://www.agilemodeling.com/artifacts/securityThreatModel.htm>.
3. "Appendix M: SDL Privacy Bug Bar (Sample)." Microsoft. N.p., n.d. Web. 16 July 2017. <https://msdn.microsoft.com/en-us/library/cc307403.aspx>.
4. "Appendix M: SDL Security Bug Bar (Sample)." Microsoft. N.p., n.d. Web. 16 July 2017. <https://msdn.microsoft.com/en-us/library/cc307404.aspx>.
5. "Secure Coding Guidelines for Java SE ." Oracle Technology Network. N.p., n.d. Web. 24 July 2017. <http://www.oracle.com/technetwork/java/seccodeguide-139067.html#5>.
6. "Enabling a Java security manager and specifying policy files for a JVM." IBM Knowledge Center. N.p., n.d. Web. 24 July 2017.