ICS 491 Assignment 2, 3, 4 & 5: SDL: Requirements and Design (Group)

Author: Spencer Luther, Jarrod Lofy, Alexandra Mink-Flacco

Date: July 15, 2016

Abstract: An online learning application targeted at younger audiences has been developed, and now account information and login security will be implemented. In order to implement these features, a report on security design for this project has been compiled. Security and privacy will need to be developed around user login and account information as detailed, and steps to ensure risk is minimized during production have been defined here. The GitHub repository for the project can be accessed here: https://github.com/lbclofy/ics491_sbsmath .

1 Requirements

1.1 Establish Security Requirements

The program should provide basic login and account privacy measures. Because the program will use a login, it will include a database of user IDs and corresponding passwords, the latter stored after hashing. Each ID must be unique; user IDs and passwords should not be accessible from the user's end, but kept server-side and separate from the program so that the user (or malicious hacker) cannot access them directly. The users will have account information (high scores) that will constitute personal data; to maintain the integrity of this data, the account information will be kept in a separate, secure database. This separate database should only be accessible by direct (physical) access by administrators, or by requests from the main program. Passwords will not be sent out from the server, only information regarding whether the inputted password matches the one in the server. All information sent to and from the server shall be secure.

Security flaws will be tracked using phase testing. When each step is completed, the existing program and program additions will be exhaustively tested to determine any existing flaws. The testing will attempt to target known security flaw categories, such as data transmission interception. Additional security/privacy threat management steps have been set up as outlined in sections 1.2 and 1.3.

1.2 Create Quality Gates/ Bug Bars

The quality gates shall require that only designated software members alter the source code, and must login to do so. Designated software members may follow the currently defined requirements, but if they desire to go off of the defined requirements, all members of the

designated software team must approve. At the end of each security implementation phase, all members of the designated software team shall review the code.

1.2.1 Bug Bars

Critical: -Spoofing of user identity: Ability to impersonate another user

-Server login information disclosure: information about a user's login and password is disclosed without permission

-Server login tampering: a user's login information is tampered with

Important: -Server account information disclosure: information about a user's high score is disclosed without permission

-Server account tampering: a user's high scores are tampered with

Moderate: -Server Denial of Service: User is unable to login due to server error

1.3 Perform Security and Privacy Risk Assessments

Upon beginning a new phase of security, the designated software development team must answer the following questions:

1. What is the current phase you are implementing?
2. Will this phase affect data transmission between the server and the interface? (Y/N)
3. Will this phase affect the method for creating a new account? (Y/N)
4. Will this phase affect the method for logging in to an existing account? (Y/N)
5. Will this phase affect the means of data storage in the server? (Y/N)
6. Who will be implementing this phase?

If the answer to any of the yes/no questions is 'yes', the software development team must perform security and privacy risk assessments, and at the end of the phase, the additions to the

program must be checked by any members of the software development team who were not primarily implementing the current phase.

2 Design

2.1 Establish Design Requirements

The app will be collecting sensitive data pertaining to each students individual performance within the app, so for obvious reasons and confidentiality our design needs to ensure that students' records are safe from outside and prying eyes. It could be as simple as jealous students or as malicious as a hacker collecting the data for whatever nefarious reasons. The design requirements are simple: as a student works through different lessons, we want to protect their progress. Nobody but the student or their teacher should be able to see how many questions a student has gotten right or wrong. Therefore, we will design a login system intended to protect the student's confidentiality. The end goal of the design is to ensure that students' performance information is safe from the outside. In order to protect students' information the design will require a registration for new users, a login for returning users, and a way to retrieve this information if it is forgotten. Additionally, the login information will have an associated password so that the login information can be protected. We don't just want anyone using any username. That would defeat the purpose of us trying to protect the students' data. This login and password is all in an effort to protect the sensitive data that is meant to be used by only the student and possibly their teacher. Additionally, we will need to make sure that there isn't any other way to access a student's info other than their login, which means we will have to prevent the app from sending information out that could be intercepted and we will also need to ensure that all of the information about a student stored on the app is safe from being taken out of the

device itself. Essentially, we will have to make sure that the only way to access a student's records is by using the login to authenticate that the correct person is using the app.

2.2 Perform Attack Surface Analysis and Reduction

The privilege levels are simple. Each student will have access to their own performance info, and a teacher will have performance info for each student in their class for the purpose of gauging their progress. After looking up exploits for similar mobile applications, it appears that the main four threats we should be looking into are as follows: poor data storage practices, unauthorized access, lack of encryption, and data leaks from syncing. This means that within our app the students' information will have to be encrypted and we will have to find a way to protect the local storage. Unauthorized access will be protected through the login and password but we will have to ensure that the password and login recovery will be robust enough to ensure that the user getting the recovery info is the right user. Lastly, we will also have to find a way to ensure that all the data is correct and protected during communications with servers.

2.3 Use Threat Modelling

Identify threats:

- Poor data storage practices
- Unauthorized access
- Lack of encryption
- Data syncing interception

Understand the threats:

- If we don't follow proper protocol for storing data, it could be relatively simple for a hacker to find the data locally. For example, we can't just store a student's info locally in

a text file. It may not be available from the app, but it wouldn't take much to be able to view that text file and take student's sensitive data.

- The main threat for unauthorized access is protecting how we send recovery info. This is precisely the reason hackers were able to get into the accounts of the leaders of social media giants. It isn't necessarily them brute forcing a password, rather it is that we have to make sure that when we send account recovery info that it is going to the right person. That said, we will have to protect against a brute force attack as well, where a hacker just guesses a bunch of passwords.
- The next threat is related to both the first and the last. Without encryption, a hacker who gets information can simply read it. But if they intercept an encrypted set of information they may still not be able to get useful or readable text from it. We need to ensure that if a hacker finds or intercepts communications it is encrypted so that it isn't just as simple as reading a file.
- Lastly, we need to make sure that all communications between the app and the servers are protected. Otherwise a hacker could intercept this information and either have the sensitive data we are trying to protect, or they could have the students exact login, which would directly give them access to the student's login info.
- For categorization of threats, please see section 1.2.1. Data flow diagram illustrated in image 1. From Image 1, please note the privilege boundaries as follows: User to Process information may be accessed by unique user or all admin, and data from processes to databanks may be accessed by all admin.

3 Implementation

3.1 Approved Tools

- All Systems: Corona SDK. Used for creating, compiling and running the program. Minimum required version: 2015 Free Subscription
- All Systems: Luacheck. Used for static analysis. Minimum required version: 0.15.1; available https://github.com/mpeterv/luacheck .
- All Systems: LuaRocks. Optionally used for installing Luacheck. Minimum required version: 2.3.0; available https://github.com/keplerproject/luarocks/wiki/Download
- All Systems/Online: GitHub. Used for collaboration when creating the program. Permitted Versions: Online website interface. Project Site: https://github.com/lbclofy/ics491_sbsmath
- All Systems: Profiler for the Corona SDK. An additional analyzer that focusses on increasing performance speed and improving memory usage. Available http://www.mydevelopersgames.com/profiler.html
- All Systems: debugger.lua. A dynamic analyzer approved for this project that is simple to implement and use. Available https://github.com/slembcke/debugger.lua

Note: You may use GitHub or Corona SDK to modify the code; if you use Corona SDK, please check out the code you are currently working on so others do not make changes while you are working on it, and update working code in a timely manner. If you use GitHub to edit code, please make sure that the code runs on Corona SDK when you are finished. In both cases, please run a luacheck on your code when you have finished.

3.2 Unsafe Lua Functions

- Unsafe User Text Entry: print()/say(), parseargs()/post() by themselves. Recommended: define a set of characters that could be dangerous, then remove them from any user input withe the function gsub(_insert characters to be removed here_). Useful for fighting cross-site scripting.
- Unsafe User Text Entry: loadstring() by itself. Recommended: use gsub(). Useful for fighting Lua code injection.
- Unsafe SQL: select()/query() sql functions. Recommended: Have prepared statements ready and call db:prepare/db:prepared. Useful for fighting SQL injections.
- Unsafe Password Hashing: MD5 & SHA1 functions/formats. Recommended: use a slow hashing function such as bcrypt, scrypt, or PBKDF2. Useful for fighting password compromising.
- Warning: include()/require()/dofile(); these allow local file inclusion attacks, but no known alternatives have been developed. Recommended: limit the use of require() and dofile() and strictly limit the use of include().

3.3 Static Analysis

We will be using the Luacheck, due to its slight security focus (which is uncommon for the few existing Lua static analysers), ease of use, and recommendations from professionals. Luacheck is an open source static analyzer for Lua, developed by Peter Melnichenko. It primarily checks for local unsafe coding practices, such as undefined global variables. As such, manual checks for specific unsafe functions will be required. Because Luacheck is open source, it is available on github at https://github.com/mpeterv/luacheck . Please see installation instructions (README.md) on the webpage for installation and use instructions.

With regards to use, installation can be a bit tricky; running is simple but does require command line knowledge. While not as convenient as working with an IDE with a built-in analyzer, it is still effective at running through a directory and pointing out errors in lua files. For example, it found several unused (exploitable) variables in lesson 1 (see image 2). Note that there are a few text editors such as Vim that allow Luacheck to display warning parallel to the source code, but these are unused. Also note that Luacheck is not an ideal static analyzer for security, but due to the lack of such Lua analyzers, Luacheck was chosen for the benefits listed above.

4 Verification

4.1 Dynamic Analysis

We will be using debugger.lua, a dynamic code analyzer developed by Scott Lembcke. While a number of dynamic analyzers touch on possible security issues, debugger.lua is the easiest to setup and use. It checks for a variety of software issues, including evaluating all functions and variables it encounters. Note that manual checks for specific unsafe functions will still be required. This software is not open-source, but is still available on GitHub at https://github.com/slembcke/debugger.lua . See README for setup instructions.

Installation for this software is very simple, and running the software is also smooth. However, it does not provide as deep an experience as I would like (though this is not out of place for a Lua dynamic analyzer). It does catch basic issues, such as pointing out a ‘mutating non-standard global’ variable called ‘system’ in the CiderDebugger file, but no high-level performance issues. The controls, however, are easy to use, and what this program does catch is easy to identify.

4.2 Fuzz Testing

Attack 1: The simplest attack, invalid login attempt. Used a user ID and password that had not been registered in an attempt to login; repeated several times. Attack failed; invalid login credentials were rebuked.

Attack 2: Attempted a Lua Injection at the login page by entering ‘; print “!” and variations thereon in both user ID and password (this would have printed the character ! if successful). Attack was attempted six times, but the attack failed, and the input was rebuked.

Attack 3: Attempted a Lua command line injection when running the program; attempted to inject commands such as activating the command “std::vector<std::string code> m_injections;” with a corresponding m_injections file in the directory. Tested several times, and each time the attack succeeded in causing an error and failing to run the program. In order to fix this, we are working on preventing command line launches of the program; forcing the user to launch with an SDK should prevent the user’s ability to launch a command line injection.

4.3 Attack Surface Review

Since the conclusion of Assignment 3, we have added a dynamic analysis tool, which should not add to the attack surface, but rather expose potential security flaws. The code itself has gone through major remodelling, with the additions of a login function and account information (high scores). Obviously, this massively increases the attack surfaces. The login interface alone opens up the possibility of Lua injections that could seriously compromise user information or cause the insertion of malware.

However, we have chosen to use a localhost for our project; this removes the threats of SQL or HTML injections, since these languages are no longer being utilized. Using the Corona SDK as the platform for running the application also reduces the capabilities of Lua injections, which would be much more effective on a command-line based Lua application.

5 Release

5.1 Incident Response Plan

In response to any breaches in the integrity of our application, we have developed a Privacy Escalation Team. Jarrod Lofy, the leader of the program implementation, will function as the escalation manager. He will immediately review reports of the damage (with the help of other team members) and determine what updates, code augmentation, server augmentation, etc. will be best to prevent any further damage. Spencer Luther, who has taken a number of law classes, will serve as legal representative in this case. He will review the damage, assess what liabilities the company and individual members may incur, and determine plans of action to protect the company and its members. Lastly, Alexandra Minc-Flacco, who has served as a mediator between team members and teachers in the past, will serve as the public relations representative. She will release statements to reaffirm the abilities of the company, and inform users about potential damage and how to mitigate this damage.

Users will be able to reach members of our team by sending an email to the address ics491secaffairs@gmail.com. Members can access the email inbox with the password JarrodSpencerAlexandra . The email was registered under Spencer Luther's name.

In the event of an incident email, the team will enact the following procedure:

1. The escalation manager will review the email and determine if more information is required from the user. If so, he will open a dialogue with the user to obtain this information.
2. The escalation manager will compile a report that summarizes a) the source of the issue, b) the impact of the issue, c) known facts about the issue, d) timeline of the issue, and e) possible plans to fix the issue.
3. The escalation manager will disseminate the report to the legal representative and the public relations representative.
4. The escalation manager shall begin working on a solution to the issue, and may enlist the help of other team members as needed.
5. At the same time as 4, the legal representative shall review the issue and create a report that summarizes the liabilities of the company and its members, and with the legal team creates a legal response plan.
6. At the same time as 4, the public relations representative shall put together a press release that details the facts of the issues while reaffirming the security of the company.
7. After 6 but still during 4, the public relations representative shall compile a list of mitigating actions users can take and send this list to all users who have the product.
8. The escalation manager shall enact the solution, distributing and installing it as necessary. If steps 5-7 are not yet complete, they shall be completed by the relevant persons.

5.2 Final Security Review

For the final security review, we have re-run all the tests described in 4.2., using more variations. We have also extensively tested every subsection of the app (every ‘lesson’) to ensure

no possible security breaches occur. After finishing this security review, we have granted the application a grade of “Passed FSR (with exceptions)”. While there are very few security issues, launching from the command line can still allow a user to enter and alter the program. In order to mitigate the potential damage for the initial release, the release shall only be available via Corona SDK; in future versions, the command line shall be secured.

5.3 Release Certification & Archive

The final release is available at https://github.com/lbclofy/ics491_sbsmath/releases/tag/f1.1

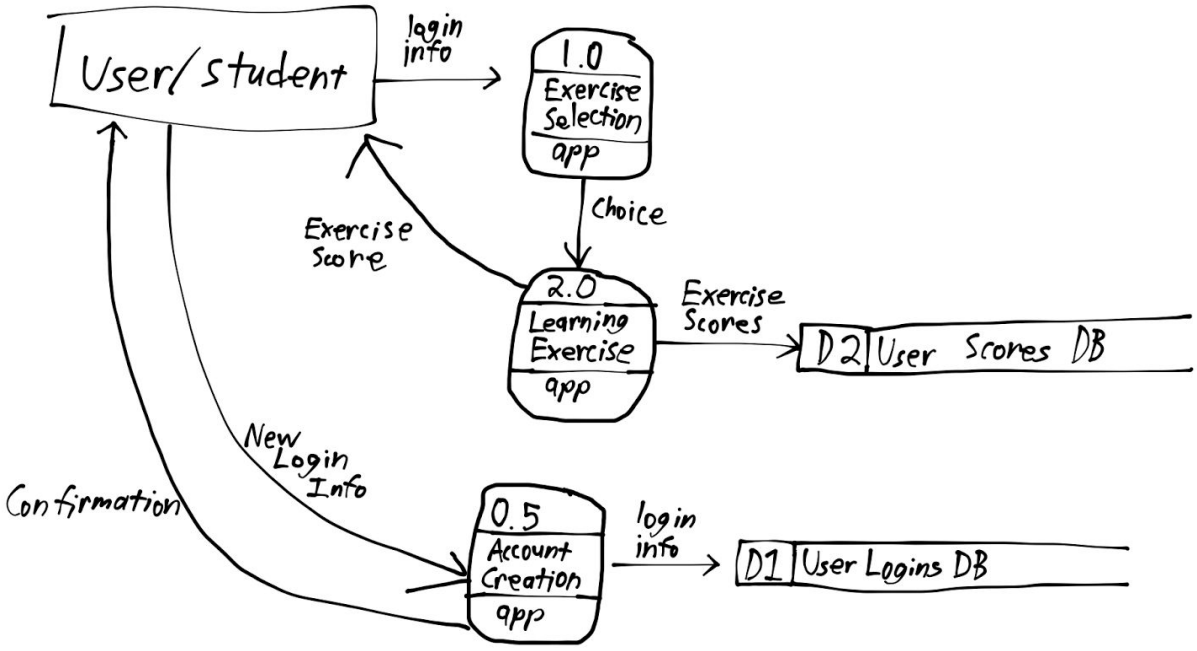

Image 1: Data Flow Diagram

```
sbs_math/lessons/lesson1.lua:3:7: unused variable animal
sbs_math/lessons/lesson1.lua:4:7: unused variable animalball
sbs_math/lessons/lesson1.lua:29:7: unused variable selText
```

Image 2: Screenshot of several errors found using Luacheck

Link to GitHub Repository for Project: https://github.com/lbclofy/ics491_sbsmath

Bibliography

1. Sunny Wear. N.D. *Module 17 Threat Modeling*. [Online]. Available: https://www.cybrary.it/video/part-1-intro-18/
2. Scott W. Ambler. 2014. *Security Threat Models: An Agile Introduction*. [Online]. Available: http://www.agilemodeling.com/artifacts/securityThreatModel.htm
3. Microsoft Corporation. N.D. *Chapter 4 Design Guidelines for Secure Web Applications*. [Online]. Available: https://msdn.microsoft.com/en-us/library/ff648647.aspx
4. Eric Beehler. N.D. *Top five mobile application vulnerabilities*. [Online]. Available: http://searchmobilecomputing.techtarget.com/tip/Top-five-mobile-application-vulnerabilities
5. Microsoft Corporation. N.D. *Appendix M: SDL Privacy Bug Bar (Sample).* [Online]. Available: https://msdn.microsoft.com/en-us/library/cc307403.aspx
6. Microsoft Corporation. N.D. *Appendix C: SDL Privacy Questionnaire*. [Online]. Available: https://msdn.microsoft.com/en-us/library/cc307393.aspx
7. Robert Harvey. March 14 2014. *What is The Process of Creating A Bug Bar*. [Online]. Available: http://programmers.stackexchange.com/questions/232425/what-is-the-process-of-creating-a-bug-bar
8. Felipe Daragon. May 26 2014. *Lua Web Application Security Vulnerabilities*. [Online].Available: http://seclists.org/fulldisclosure/2014/May/128

9. Microsoft Corporation. N.D. *Appendix E: Required and Recommended Compilers, Tools, and Options for All Platforms*. [Online]. Available: https://msdn.microsoft.com/library/cc307395.aspx
10. Microsoft Corporation. N.D. *Appendix K: SDL Privacy Escalation Response Framework (Sample)*. [Online]. Available: https://msdn.microsoft.com/library/cc307401.aspx
11. Microsoft Corporation. N.D. *Phase Five: Release*. [Online]. Available: https://msdn.microsoft.com/en-us/library/windows/desktop/cc307420.aspx#EB
12. Various, Anonymous. July 2, 2016. *Corona (software)*. [Online]. Available: https://en.wikipedia.org/wiki/Corona_(software)
13. Debbie Fletcher. March 30, 2016. *Static Code Analysis Tools for Bulletproof Software Security*. [Online]. Available: http://www.infosecurity-magazine.com/opinions/static-code-analysis-tools/
14. Peter Melnichenko. N.D. *Luacheck*. [Online]. Available: https://github.com/mpeterv/luacheck
15. Daniel Gruno. N.D. *Database Connectivity*. [Online]. Available: http://www.modlua.org/api/database#database_caveat
16. Scott Lembke. 2016. *Debugger.lua*. [Online]. Available: https://github.com/slembcke/debugger.lua .
17. Anonymous. December 10 2015. *Debugging Lua Code*. [Online]. Available: http://lua-users.org/wiki/DebuggingLuaCode
18. M_D_K. February 5 2009. *Lua Code Injection*. [Online]. Available: https://medek.wordpress.com/2009/02/05/lua-code-injection/