Team Alpha

Mark Arakaki

Blake Larson

Hendricks Hicks

Jesus Gonzalez

ICS 491

Professor Barbara Endicott

August 6, 2016

Assignment 5: SDL: Release

The application our team plans to develop is a web based password manager that stores the information in a remote database i.e. not on a user's device. The application will be locked behind a two-step authentication system mixed with a password and a number of security questions that will be set by the user. For the development to run smoothly we'll be following the Security Development Lifecycle (SDL) provided by Microsoft to ensure that our application is safe not only post-development, but during development before the release. To provide assurance of the safety and privacy of our application we performed steps one through seven of the SD Process. Our team established security requirements, created quality gates/bug bars, performed security and privacy risk assessments, established design requirements, performed attack surface analysis/reduction, and used threat modeling. Our team has a firm understanding that following the requirements and design stages of the SDL process will ensure that throughout the development lifecycle of our web based password manager will see improvements in regards to security, privacy, and costly issues. The first portion of the SDL process that we established are the security requirements.

Establish Security Requirements:

The password manager application has a multitude of areas in which privacy will be a concern for users. As developers it's crucial to allow complete transparency in terms of what information is being collected and how it is being used. Any data that is sourced from the user must be handled with care and integrity. Ensuring that the data is not only protected but also used only for the reasons provided to the user is crucial in maintaining a healthy relationship between the developer and user base. We must also ensure that all handling of data is in compliance with federal, state, and international laws. The application being created will be collecting passwords, usernames, and website information. We must verify that the information collected does not reach beyond the point of personally identifiable information (PII) and into sensitive PII. In terms of privacy requirements it is important that the application only collects data that is required to successfully perform the task that the app was created for. This application will not store names, addresses, or any further information of the users. Finally, it will be necessary to receive consent from the user whenever their data is being processed in order to fully keep the user aware of what is happening with their inputted data.

Furthermore, securing the information being stored within the application's database is of the utmost importance. The application must have a system in which the user can create an account within the application to store his/her websites, usernames, and passwords. He/she will also create a password, custom security questions, and answers that he/she must answer when attempting to login to the application to gain access to the database of account information. This two-step authentication process will prevent unauthorized access to the account's information.

These are the security measures that will be taken when developing this web based password manager.

Keeping track of security flaws during the development will be done with a policy that emphasizes commonly scheduled tests that will outline new and existing vulnerabilities. After the discovery of both old and new vulnerabilities we will then analyze and ranked by a threat level so that as developers we address the most dangerous security flaws first before moving forward. The analysis that takes place will also help to indicate the causes of the vulnerability so that further in the development cycle the team does not run into the same problem. The key to our team's tracking of security flaws is consistency and dedication to not only discovering and solving the security flaws, but analyzing them thoroughly to fully understand how the problem originated in the first place. This is how our development team will keep track of security flaws in the development cycle of the web based password manager.

Create Quality Gates/Bug Bars:

The development cycle of the web based password manager must begin with quality gates/bug bars that will assist in fully understanding the risks associated with this specific application. These quality gates/bug bars will also ensure that we can successfully identify and construct a solution for security bugs throughout the development cycle. The quality gates/bug bar for this application's development will be divided into a privacy bug bar and a security bug bar. In terms of the privacy bug bar there are four scenarios that must be accounted for throughout the development cycle.

Privacy Bug Bar
• Lack of notice and consent

- Lack of data protection
- Lack of internal data management and control
- Data minimization

First of all, the lack of consent vulnerability is vastly important due to the development's team goal of transparency between the user and the handling of the data inputted by the individual. All information needed to be entered into the application by the user must be verified with user consent and any process that the data is going through must be thoroughly explained to the user and approved. Secondly, the privacy vulnerability of lack of data protection is a huge point of emphasis when creating an application that is created to safely store information such as website usernames and their corresponding passwords in a database. Security authentication must be a constant and ever growing part of the development cycle to ensure that this vulnerability never rises to uncomfortable levels. The lack of internal data management and control is also a vulnerability that needs to be accounted for. We need to ensure that we construct a retention policy for the data that is being stored within the application. The deletion of data within the application will only be directed by the user to ensure that they are fully understanding that their passwords and usernames of various websites will be wiped clean from the application's memory. Finally, data minimization needs to be a factor when programming the application. Only asking for pertinent information that will assist the application in fulfilling its purpose is crucial to the privacy of the user. Furthermore, it is important to also create a security bug bar to help indicate the various vulnerabilities that can affect it.

Security Bug Bar
● Elevation of privilege ● Denial of Service ● Tampering ● Spoofing

- Information disclosure
- Security features
- Security assurances

In addition, having a security bug bar on an application that stores data on a database is equally as important before the development cycle begins. The elevation of privilege vulnerability should be noted when we develop ways for an administrator to handle the data stored in the application's database. This vulnerability could be highly problematic if individuals discover a way to bypass the application's two-step authentication and are able to see various user accounts and the information that is stored on the individual's account. We must also ensure that denial of service attacks can be stopped due to the ease in which an attacker can perform them to our servers. This type of attack has become commonplace in today's cyber security world and we must also take steps to ensure that it does not affect the security and privacy of our application's users. We must also be aware of tampering that may occur with our software. We must ensure that our database that holds confidential user information does not become vulnerable to this type of attack. Furthermore, spoofing is another problem that may occur with individuals attempting to be identified as a certain user of the application to gain access to the user's information. Another vulnerability that must be known is information disclosure. The development team needs to be aware that information stored within the application should not be able to be viewed from anywhere on the system other than the intended area in which the application was designed to display it. This also includes leaks of random heap memory that may occur from careless designs in the construction of the application's code.

Another vulnerability that needs to be addressed is that our team needs to create close to bulletproof security features that will ensure that the data hidden behind the application's authentication methods are safe and secure. Awareness when it comes to our programmed

security features are not only necessary for the applications integrity but to uphold trust between the user and the application. Finally, security assurances must be addressed because the application must offer the users a feature/function that they can fully trust to protect their information. We will also need to provide various security bulletins or security information to the users to keep them informed and aware of various security features that are available to them. Having quality gates are crucial to have in a development's team possession before the development cycle occurs due to its ability to help understand, identify, and solve privacy and security risks that most likely will arise throughout the development cycle of the application.

Perform Security and Privacy Risk Assessments:

In order for our software to be as secure as possible, we need to first identify the privacy impact rating (PIR). This rating represents the degree of risk our program presents from the perspective of privacy for the user. Since the program's main intent is as a password manager, which utilizes a database to send and retrieve the requested data; we can give it a rating of P3. Since the data (Personally Identifiable Information) is not locally stored on the user's computer nor is it being transferred from it, the rating is no longer considered a P1. The portion of our software that is of top concern would be the initial login page. This will need a threat model as it's the basic safeguard for the specific user's data. Looking into the login screen a bit deeper, the question of a user forgetting their password arises, which they will need to provide their email to send a password reset form. This form as well as the login screen could cause security issues, in which both would need threat modelings and be reviewed extensively. If we look at a potential attacker's standpoint, they may try to breach the login page via unauthorized access or trying to brute force access; we can work around this by issuing a timeout period for the username or IP address. This would stop infinite amounts of login spam as well as security breaches.

Establish Design Requirements:

In order to meet the development team's goal of transparency, the user will be asked to agree to the application's user agreement, which will outline the user's data that is stored, where it is stored, and how. Anytime one of these factors is changed the user will be asked to consent to the changes via a pop up window.

To ensure the integrity and safety of the user's data, the development team should use a secure connection to communicate with users to prevent information disclosure. The user must be authenticated before gaining access to any stored files, this access must also be localized in order to prevent accidental information disclosure and tampering. The user information must be stored in a safe and protected space, where the user has confidence it will not be easily accessed by unauthorized users. We will then encrypt this data with the user's newly created password so that unauthorized users will not be able to read the stored data without an the encryption code. However, the user input will not be used to hash the location of the user data. This can possibly lead to unauthorized access. Once the data is accessed the user will need a final password that is unknown to the development team, in order to read their user file.

Perform Attack Surface Analysis/Reduction:

Considering that our application is meant to be used remotely, we don't have to worry about client operating systems and their devices, but instead our infrastructure that it's running on. A big benefit of this is that we can be very strict with our configurations and don't have to worry about privilege escalation in the sense of an attacker leveraging our software on a device. But what we do have to worry about is that we'll be storing critical information and makes us an attractive target. One of the most sensationalized attacks are distributed denial of service.

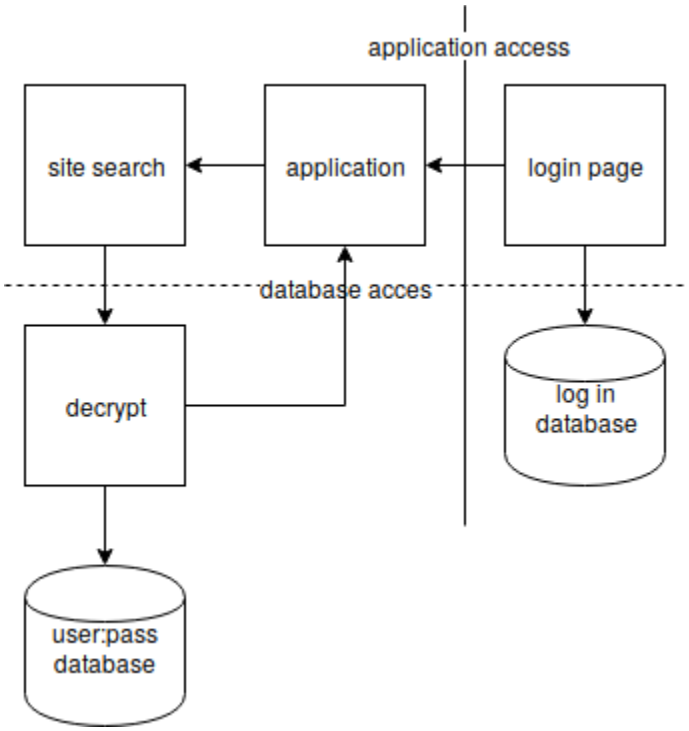
Because of how much it's grown in complexity in the past few years, it's something that can't easily be defended against, but it could be mitigated with Amazon web service's automatic scaling cloud services.

Since this is on the open internet we'll be open to any and every type of attack. At this point, we'll have to consider not only application security, but also cyber security. When it comes to compromising a web server, the attack vectors list is a few miles long, but we can't talk about the app without talking about how it's being provided. SQL injection, Java injection, remote exploitation, social engineering, and out of date software are a few common vectors. And since our service is the 'master key' it would allow an adversary access to other passwords, so we have to consider almost every and any type of attack. If this were a side/class project, we'd offload our cyber security responsibilities to our 3rd party hosting provider. But if this is something we're to take care of ourselves, the easiest mitigation is network segmentation, keeping software up-to-date, and of course ensuring we have solid procedures and policies and that our 'staff' are properly trained and familiar with them. Another simple mitigation would be to encrypt the stored information. Trying to keep an attacker off of our systems might not be very simple, but if they do gain access, they won't be able to use any of the information they gather. As for application attack vectors, they're somewhat limited. Since we're essentially just storing text after a person logs in, we don't have to worry about complex interaction in our software. We'd only have to worry about user input since a user logs in, then they're shown their passwords.

Use Threat Modeling:

To explain the threat model a bit, we've decided that separating the login database from the password storage would prevent it needed to be access unnecessarily. Since the login page

and it databases will take a beating from the wide internet, they're at the very edge in their own area. But a key feature is that the application itself doesn't have access. The application makes a request to a 'Search' agent, who then request the 'Decrypt' agent to return the user:pass pair to the application to display. This set up prevent direct user request to the databases and give us good segmentation.



In conclusion, our team has implemented the steps one through seven of the SDL process to improve security, privacy, and to reduce costly issues that may plague the application's development cycle. Our team has successfully established security requirements, created quality gates/bug bars, performed security and privacy risk assessments, established design requirements, performed attack surface analysis/reduction, and used threat modeling. We believe that a mixture of these steps provided by Microsoft will give us an advantage in terms of security and privacy not only throughout the development cycle of our application, but post launch as well. The importance of having a standard set before development even begins is crucial for

quality control and having a vision of what the team envisions the app to be when finished. There are still other steps to be performed in regards to the SDL process, however we have taken the first steps in creating a foundation of which our applications success will be built upon.

The area below will describe the implementation phase of our applet's development. This phase is dedicated to constructing blueprints that will lead to the successful development of our website password bank applet. This step is crucial in assigning which tools that our development team will be using throughout the development cycle. Our team also discussed and determined what unsafe functions will be prohibited from developers on our team from using. We also discussed alternate functions that perform similar tasks without the flaw(s). The implementation phase is yet another step that will help us build a solid foundation so that when we begin programming our applet, we have a set standard and toolbox to work with.

Implementation Phase:

The area below will describe the implementation phase of our applet's development. This phase is dedicated to constructing blueprints that will lead to the successful development of our website password bank applet. This step is crucial in assigning which tools our development team will be using throughout the development cycle. Our team also discussed and determined what unsafe functions will be prohibited from developers on our team from using. We also discussed alternate functions that perform similar tasks without the flaw(s). The implementation phase is yet another step that will help us build a solid foundation so that when we begin programming our applet, we have a set standard and toolbox to work with.

To develop our website password bank application we compiled a list of tools that we are planning on using during the applet's development. The tools that our development team used

are listed below along with the minimum version of the tool that we used and why we decided to use said tool:

Compiler/Tool	**Minimum Required Version and Switches/Options**	**Optimal/Recommended Version and Switch/Options**	**Comments**
Java Compiler	JavaSE-1.8 Developer Tool Kit		We are using this developer kit to compile our Java code.
Static Analysis Tool	Checkstyle 8.0		Dev team used this to assist in inspecting our Java code so it adheres to our defined coding rules
Applet Components	JavaSE-1.8 Developer Tool Kit (Swing)		Within the toolkit we used Swing to create our components

Deprecated Java functions are notated by @Deprecated, in which programmers like ourselves are highly discouraged from using. These functions can cause security risks and be a risk to use, or a better alternative exists. In the front end of our application, we utilize many GUI based functions from the Java libraries, but so far functions like setLabel(), show() are deprecated and needing some alternatives in their place. The function setLabel was replaced by setText(text), where it immediately changes the button's text. As a precautionary change the name of the function no longer states label, because it could be confused with creating an actual label rather than just changing the text on the label button. The function show(), shows or hides the attached component based on the value within it. As a replacement for show(), show(b), hide(), or hide(b), we just use the function setVisible(b). If the boolean b is false then it will hide the value of the component, and vice versa. As we move more into the backend development of

our application, we will start using more secure methods of integrity and most likely use functions that are replacements for deprecated ones that fall under security risk.

For our static analysis during the development cycle we decided to use software called Checkstyle. The reasoning behind this is that we could modify the coding standards that the software will be checking so we can specifically pinpoint rules that we want followed within our code. We all have prior experience with Checkstyle due to participating in a previous ICS course so we didn't run into a lot of trouble when using this software. The software is extremely straight forward and easily customizable to our privacy, security, and efficiency needs. So far, our team has experienced success with the software through a few specific examples. One of the successes that we experienced was when we misspell certain variables throughout the program and the names lack consistency. Checkstyle points out when a variable has not been initiated and allows our developers to swiftly fix our error. Another specific example of a Checkstyle success is when we sometimes get mixed up about what variables are being inherited and/or extended and Checkstyle catches it. With our multiple classes this could have become very problematic, but Checkstyle was able to catch our errors so that we could fix it without running tests for hours. Static analysis through the Checkstyle software is providing our development team with great assistance in locating our errors in security, privacy, and efficiency.

In conclusion, our development team began the implementation phase to give us a solid foundation in which we can build our website password bank applet. We first created a list of compilers and tools that we will be using throughout the development cycle. Our team then decided what deprecated functions we would need to replace with more safe and secure functions. Lastly, we implemented a static analysis tool called Checkstyle to assist us in our programming to improve security, privacy, and efficiency in our code. The implementation

phase has created a foundation of tools that we can look back to during the development cycle to assist us in creating the safest and most secure code possible.

Verification Phase:

The verification section of the report will discuss a variety of topics including the use of dynamic analysis tools, fuzz testing, and our findings from our attack surface review. As a team decided to use the software JUnit to help with testing our applet. We also performed fuzz testing to our applet with a variety of different techniques in an effort to better secure our applet for release. Our team also looked at the tools that we approved during our implementation phase and ensured that we were aware of any updates or changes to the software and how they could impact our project moving forward. The verification phase is another key step in ensuring that our applet is not only fully functional, but also secure.

Dynamic Testing

We as a team are most familiar and comfortable with using JUnit testing analysis. Jesus and I are both utilizing unit testing, as we add more and more functionality to the code. Since our application runs off mainly user input to the applet, we are testing at the ground level right now. This meaning, running the program then entering input to see if the program will crash. As we are not yet able to implement full database usage throughout the course of the application runtime, we are testing at a minimum right now. For future testing, we will create an entirely new class that will implement not only unit testing, but also regression testing to make sure that our program will run properly through and through even with new code. Specific testing includes but are not limited to: login data and non-case sensitive entries, proper database storage of new data (insert, select, delete statements), secure authorization for users (one user allowed at any given time), privileges are set (normal users can’t delete entire existing tables or data from

others). These are only some of the testings we are looking forward to implementing once our database connection is fully developed and running properly.

Fuzz Testing

At this time we could not perform fuzz testing due to our application's database not properly functioning to this point. We are currently working on the code and should be able to have the applet up and running soon. When the applet is successfully connected to its database we will then perform fuzz testing along with security improvements based on the information that we receive from the internal fuzz testing.

Attack Surface Review

During the implementation phase we discovered that we mislabeled the JavaSE-8 1.8.0 as JavaSE-1.8. Our development team has been using the current build of Oracle's JavaSE Developer Tool kit that provides us with a multitude of features that increase the security and efficiency of the code that we create. We also added JUnit 4.12 to the list of tools that we have used for this applet's development. This mislabel was a major typo error that occurred when writing the report and we have now relabeled the tools that we are using for the development of the project as follows:

Compiler/Tool	**Minimum Required Version and Switches/Options**	**Optimal/Recommended Version and Switch/Options**	**Comments**
Java Compiler	JavaSE-8 1.8.0 Developer Tool Kit		We are using this developer kit to compile our Java code.
Static Analysis Tool	Checkstyle 8.0		Dev team used this to assist in inspecting our Java code so it adheres to our defined coding rules.

Applet Components	JavaSE-8 1.8.0 Developer Tool Kit (Swing)		Within the toolkit we used Swing to create our components.
JUnit	JUnit 4.12		For testing and fuzzing purposes, we are running unit and regression methods.

In regards of the typo nothing has changed in terms of new versions of both the JavaSE-8 1.8.0 Developer Tool Kit and Checkstyle 8.0. In the span of a few days no major patches, updates, or vulnerabilities have been discovered. This is extremely comforting for our development team due to this consistency provided by the software tools that we are using not being tampered with by way of security flaw or software updates. Overall, in terms of the tools that we are using for the development of our website password bank applet we are using the same tools as last week with the exception the JUnit 4.12 software.

In conclusion, our team has taken the appropriate steps to ensure that the verification phase of our applet's development cycle is a success. We performed dynamic analysis using JUnit which helped us to test out our applet so that we were receiving the results that we were looking for. However, we couldn't perform the fuzz testing due to our applet not being able to connect to our database properly yet. This fuzz testing should be performed at a later time when the applet is appropriately connected to the database, along with security measures applied as well. As for the attack surface review, the list of tools has been updated and the typo from the implementation phase has been fixed. Other than the fuzz testing, we have taken the necessary steps to ensure that the flow of our team's development cycle on our website password bank is smooth and successful.

Release Phase:

Our group has moved into the release phase of our website password bank applet which entails a creation of a multitude of policies for release and post-release. The first plan that we have created for the release of our applet is an incident response plan that will address when privacy and other types of incidents occurs post-release. This phase also includes a final security review that will fuse together our team's threat model, static and dynamic tools, and quality gates/bug bars to analyze and enhance the security of our applet. Lastly, our team will ensure that all documentation is in order for the applet's release. This phase is the final step that will culminate in the successful release of team alpha's website password bank.

Incident Response Plan:

First of all, our team took a number of steps when creating our incident response plan. As a team, we decided that Mark will be the escalation manager. Mark will be responsible for including appropriate representation throughout our team and driving the process to completion. It was also decided that Blake will be our applet's legal representative and will handle and concerns relating to the legality of our applet. Hendricks has been assigned the responsibility of PR representative and will handle any public relation incident or concern that comes from the existence of our applet. The security engineer position will be given to Jesus and his responsibilities cover a multitude of aspects pertaining to the maintenance and upgrading of our applet to ensure that our applet is up-to-date with security features while also maintaining it's integrity. If a user experiences any problems that seem to be extremely problematic they can reach our privacy escalation team at marka2@hawaii.edu. This e-mail is the best way to get into direct contact with all members of the team immediately. Our team will then attempt to diagnose the problem and assist the user to the best of our ability.

Our privacy escalation team will follow a certain process when an incident does occur. The first step that our team will take is investigating the source of the incident. By having a clear vision of what the source of the incident is we can then gain an understanding of the impact and breadth of the escalation. After we have an understanding of the source and impact of this incident we can then investigate the validity of the incident. This step is incredibly important because as a privacy escalation team we must only allocate our time and resources towards incidents that are valid. If the information of the incident is valid then as a team we can collect the known information and create a summary of what we know. This summary of information will then give our team the ability to create a timeline of expectations in regards to finding a solution to this incident. Mark, the escalation manager, will then assign portions of the solution workload to the appropriate members of the team. Mark will ensure that the incident does not escalate and a solution is found in a timely manner. This is the process that will occur when an incident arises and is brought to our attention.

Privacy Escalation Response Process
1. Find the Source of the Escalation
2) Investigating the Impact and Breadth of the Escalation
3) Determining the Validity of the Incident or Situation
4) Create a Summary of the Known Facts
5) Create a Timeline of Expectations
6) Assign Employees to Various Sections of the Problem

Final Security Review:

As end users for the application the team created, we have come to a formidable conclusion that the grade should be “Passed FSR with Exceptions”. We have agreed upon this

grade due primarily to the fact that issues the team ran into during development, caused revisions of certain parts of formatting. Specifically, the team dealt with server problems and needed to start more basic. We started anew and built the program solely on a single user basis. The user who runs the program on their localhost can create whatever database with their own specific credentials. On a more secure side of the development, we have created a login page that is specific to the user's choosing. These credentials will give access to all the data owned by the user and allow them to add more tuples. To fix know credential connection issues to the database, the team needed to develop a second way of connecting to the localhost database. This is the largest security risk until the team can develop a better way of clearing loose credentials in the code. In terms of solidity and rigidness, the application works very well at keeping improper users out and the localhost in. As for the grade that the team came to an agreement on, we are certain that the known security issues can be resolved as the bigger problems have been solved in the current outlook.

Certified Release & Archive:

Please follow this link to our GitHub to view the applet's most up-to-date README.md file and other relevant documentation for the applet:

https://github.com/marka2/ics491passwordbank

In conclusion, our team has finished the release phase of the SDL and our website password bank applet has been made available through our team GitHub web page. For our team to complete this phase we created an incident response plan that included assigning responsibilities within a privacy escalation team and privacy escalation response process that will be followed when an incident related to our applet is brought to our attention. This along with contact information is made available to the users to contact the team and get an immediate

response. We have also performed our final security review to ensure that our applet fulfills all our security requirements. Finally, we made sure to have all documentation for our applet ready and available for public consumption, including a link to the project's GitHub. This concludes the release phase of the SDL for our website password bank.