“FOR YOUTH FOR LIFE” AN ONLINE EDUCATION SYSTEM USABILITY AND SECURITY IMPROVEMENT

by

Nayana Teja Talluri

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Masters of Science

Auburn, Alabama
May 10, 2015

Keywords: HP Fortify, Secure Code Review, Online Learning, Security Risks, SQL Injection, Privacy Violation

Approved by

Cheryl D. Seals, Chair, Associate Professor of Computer Science and Software Engineering
Dean Hendrix, Associate Professor of Computer Science and Software Engineering
James Cross, Professor of Computer Science and Software Engineering

Abstract

The FYFL (For Youth For Life) application is used to perform website functions which helps learning among communities across many content areas, supports the 4-H badge management system and creates users of different levels. The 4-H Digital Badging System is a web-based learning environment providing 4-H staff and instructors with an on-line platform to create courses, activities to approve and to issue proposed badges with more convenience and efficiency and to create a fun learning experience. 4-H in the United States is a youth organization administered by the National Institute of Food and Agriculture of the United States Department of Agriculture (USDA). This online learning application's functions include forms that collect personal, classified and confidential information from different populations of different levels. A need arises to protect the privacy and security of personally identifiable information from hackers by assessing and improving the security of the FYFL application periodically.

With the growing availability of the Internet, a number of diverse user devices facilitate the demands of online learning. Online learning brings with it all the security risks inherent to using the Internet. The consequences of a security breach are loss of revenues, damage to credibility, and loss of customer trust. Security is essential as a means to retain users' trust in the online learning environment as any risk can affect students' perceptions of a systems' reliability and trustworthiness.

Table of Contents

List of Figures

1. Introduction

1.1 Motivation

An application can have great features such as a firewall, passwords systems etc., but if the rest of the system is not secured, attackers will drive right around the application's security. It used to be that a website would just display data "Here's where our store is, here's what we sell", but today, websites also take data from users: usernames, date of birth, email id etc. To retrieve this data, there needs to be a hole in the firewall. In other words, that hole is there intentionally, because we want to get data from our users. Developers tried to secure that hole in several ways. SSL (Secure Socket Layer) creates a secure pipe to the user, but if that user is malicious, SSL gives zero security. There is also a focus on automated systems such as intrusion detection, intrusion prevention, and app firewalls. To that end, this research will focus on the way to secure an application by programming the application to securely handle malicious data.

1.2 General Area of Research

Computer security has experienced important fundamental changes, over the past decade. Traditional approaches to computer security focused almost exclusively on the network. The idea was to keep malicious hackers away from vulnerable machines by placing a barrier around a local area network and the Internet. Researchers say, although firewalls certainly have their place in computer security and have since become ubiquitous, serious security problems persist.

Researchers consider code review and architectural risk analysis to be the security best practices. Among these two best practices, code review with a static analysis tool is the easiest

and most straightforward to adopt. The code review has been partially automated with sophisticated tools. Static analysis identifies many common coding problems automatically before a program is released.

The Secure Source Code review is an activity of the development phase in the Secure Development Lifecycle (SDL) and is the process of reviewing the source code of an application to verify that security controls are present, that they work as intended, and that they have been invoked in all the right places. It helps to identify any deviations from security requirements or security best practices as well as to identify security vulnerabilities in the source code. Also, the study includes an analysis of the major security risks in code such as Cross-Site Request Forgery (CSRF), Cross Site Scripting (XSS), Password Management, Privacy Violation, SQL Injection etc.

1.3 Purpose of Research

The project aims to provide security for the online learning environment. Security is essential as a means to retain users' trust in the online learning environment as any risk can affect students' perceptions of a systems' reliability and trustworthiness. The growing availability of the Internet and the number of diverse and user devices facilitate the demands of online learning. Online learning brings with it all the security risks inherent to using the Internet. Online learning is built on trust, information exchange and discussion.

The consequences of a security breach are loss of revenues, damage to credibility, and loss of customer trust. As a result, it is crucial to identify the underlying factors that can cause security issues in online learning.

1.4 Approaches to the Research

This study will focus on application security, security testing and techniques for security testing. It also focuses on choosing one among the various commercial and non-commercial tools to perform a secure code review. The main criterion to choose a tool is to identify the security vulnerabilities existing in the FYFL system.

The focus of this study is to improve the security of the FYFL system. In the first phase, we fix the vulnerabilities that were identified during the initial code scan by implementing whitelist data validation techniques. The study will create an environment to leverage existing technology of automated tools and utilize this in identifying the risks. In the second phase, we re-scan the modified code to see if the issues are eliminated. The results of this study will be used to support the methodology implemented to mitigate the risks involved. Our aim is to gain the customer's trust by securing their credentials.

2. Statement of the Problem

2.1 Research Problem

Due to development of the Internet, more and more people are taking online courses. The FYFL system provides numerous online courses to allow 4-H members to increase their competency and to upgrade their skills. Online learning depends on the Internet for its execution. However, there are many illegal activities and security threats taking place on the Internet. Consequently, the environment is inevitably exposed to constant security threats, risks and attacks. In online learning, security is basically when the learning resources are available and unimpaired to all authorized users when they are needed. Every element in the FYFL system can be a potential target of hacking or attacks, which may lead to unauthorized modifications or destruction of educational assets.

2.2 Research Approach

The main goal of this study is to protect the FYFL system from identity theft, impersonation, and inadequate authentication. The risk is great as the functionalities and features of the FYFL system become more complex and is increasingly exposed to security threats.

The study's main target is to give a detailed description of the existing FYFL system from a security standpoint. It elaborates on technologies used to build the system. The study will focus on the usability of the security tools and by doing a comparative study of its usefulness. A complete analysis of a comprehensive study on web application security risks will be discussed.

Furthermore, it describes the methodology to find the security issues and fix them to improve the security of the "For Youth For Life" system in order to resist possible attacks like SQL Injection, Cross Site Scripting (XSS), Cross Site Request Forgery (CSRF), Privacy violation etc.

2.3 Research Questions

In the first phase of the study, we plan to investigate the security aspect of the FYFL system using a secure code review automated tool "HP Fortify on Demand". In the next chapter, findings will lead to a comprehensive focus on the security issues detected in the code and improving the security of the application.

Research questions were generated to confirm and justify the theory above:

1. What are the key techniques to be implemented while programming an application to prevent the security vulnerabilities?
2. Is whitelist validation better than blacklist validation?
3. Is the secure code review method more efficient than other security testing methods?
4. What are the major vulnerabilities currently existing in the FYFL system?
5. Will the mitigation techniques implemented eliminate the vulnerabilities and securely handle the malicious data?

2.4 Hypothesis

Hypothesis I

$H0_1$: The system is not secure against possible attacks like SQL injection, Cross Site Scripting, Cross Site Request Forgery etc.

HA_1: The system is secure against possible attacks like SQL injection, Cross Site Scripting, Cross Site Request Forgery etc.

Hypothesis II

$H0_2$: There is no difference in users' preference when using a securely improved site compared to the original site.

HA_2: There is difference in users' preference when using a securely improved site compared to the original site.

Hypothesis III

$H0_3$: There is no difference in the magnitude of learners' trust when using the securely improved site compared to the original site.

HA_3: There is a difference in the magnitude of learners' trust when using the securely improved site compared to the original site.

Hypothesis IV

$H0_4$: Whitelist validation is not the best way to validate the input compared to blacklist validation

HA_4: Whitelist validation is the best way to validate the input compared to blacklist validation

2.5 Experiment

With the emergence of online environments, many organizations have resorted to automated security testing tools. However, every organization has expertise in different technological skills and varied access to technology based upon the location and focus. Due to

the variety of automated testing tools available, there is a debate on which tool will be more suitable for the organization to easily improve the security of the applications.

Though there are many ways to perform security testing, many of them have not fully ascribed to the available best practices. We choose “Fortify on Demand”, a managed application security testing service provided by the HP fortify which enables organizations to quickly test the application security of a few applications or launch a comprehensive security program without additional investment in software and personnel.

In this experiment, we performed a static analysis of the FYFL system code through the tool Fortify on Demand. Our methodology involves fixing certain issues in the code and re-scanning the code to see if the methods used to improve the security eliminated the issues. The benchmark will be the current system code; the version that we will be comparing for improvement will the next version of this system with added functionality.

2.5.1 Setup of Experiment

To perform the experimental tasks, participants must have access to the Internet through a web browser. The specifications of the machines will not be considered, but the latest browsers are recommended. The system can also be accessed through hand held devices, iPad, iPod, web accessible cell phones etc. The experiment criteria includes Science, Engineering and Technology (SET) abilities, Application Security and Security Testing knowledge, Performance on key learning activities, and Programming skills.

2.5.2 Experimental Procedure

This chapter deals with the details of implementation for modifying the code of the FYFL system after the analysis done in the initial phase. Phase I of the research basically includes

gathering the core and functional requirements. Furthermore, Phase II describes the system framework and methodology to eliminate the security vulnerabilities like cross site scripting, SQL injection. Also, the research outlines the use case diagrams and the activity diagrams which depicts the system requirements. Afterwards, the results of the automated tool before and after adding whitelist input validations to mitigate the risks form the outcome of the study.

3. Literature Review

3.1 Usability

3.1.1 What is Usability?

Usability is considered to be one of the most important quality factors for Web applications along with other factors such as reliability and security. Web application usability is not just about a good interface design. As the visual appeal of a web application is important, the fundamental structure and programming can also contribute to "good usability" [10].

Usability can be defined as “the extent to which a product can be used by certain users in a specific context of use to achieve the specified objectives effectively, efficiently and satisfactorily”. This definition offered by the international standard ISO/ IEC 9241-41 describes the most important issues to be taken into consideration when designing an application: Who are the users of the application? What are the objectives of the application? In which context is the application used? Can the objectives be achieved effectively, efficiently, satisfactorily? Satisfactory means that the application can be used without any problems and has resulted in a positive user attitude towards the application [1].

3.1.2 Usability Evaluation

The ease or difficulty that users experience with these Web applications determines their success or failure. Usability Evaluation Methods (UEMs) which are specifically crafted for the Web, and technologies that support the usability design process, have therefore become critical [1].

The most frequently used type of UEM is *user testing* [1]. This indicates that most evaluations are performed during the later stages of the web development life cycle. The following are some of the sub-types of user testing methods:

1. *Think-Aloud Protocol*: User thinks aloud while performing a set of specified tasks
2. *Question-Asking Protocol*: Testers ask users direct questions
3. *Performance Measurement*: Testers or software tools record usage data and obtain statistics during the test.
4. *Log Analysis*: Testers or Software tools analyze the usage data. For example, when usage data is particularly related to gaze points obtained from the analysis of eye movement, the method is called eye tracking.
5. *Remote Testing*: Testers and users are not co-located during the test. This method can be applied in conjunction with log analysis methods.

Inspection methods are intended to be performed by expert evaluators. Most of them are web designers [1]. The following are some of the sub-types of inspection methods:

1. *Heuristic Evaluation*: Expert evaluators identify heuristic violations in web artifacts
2. *Cognitive Walkthrough*: Experts simulate a user's goal achievement by going through a set of tasks.
3. *Perspective Based Inspection*: Experts conduct an oriented and narrow evaluation that can be based on design perspective, inspectors' tasks, and metric calculation.
4. *Guideline Review*: Experts verify the consistency of web artifacts by using a set of usability guidelines.

Inquiry methods focus on gathering subjective data from users. Most of these methods are used in combination with other methods such as testing and inspection methods to perform a complete evaluation [1]. The following are some of the sub-types of inquiry methods:

1. *Questionnaire*: Users provide answers to specific questions.
2. *Interviews*: One user and one expert participate in a discussion session concerning the user's attitude towards the artifact to be evaluated.
3. *Focus Group*: Multiple users participate in a discussion session concerning their attitudes towards the artifact to be evaluated.

Analytical Modeling focuses on modeling certain aspects such as user interfaces, task environments or user performance in order to predict usability [1]. The following are some of the sub-types of Analytical modeling methods:

1. *Cognitive Task Analysis*: User tasks are modeled in order to predict usability problems.
2. *Task Environment Analysis*: Evaluation of the mapping between users' goals and user interface tasks.
3. *GOMS Analysis*: Human task performance is modeled in terms of Goals, Operators, Methods, and Selection rules (GOMS) in order to predict execution and learning time.

3.2 User Interface Design

3.2.1 What is User Interface Design?

User interface design constructs the medium of communication between humans and applications. If the user has to think too much, that designer is not doing a good job. Good user interfaces let users complete their goals. There is a lot of up-front learning that the designer needs to do before he starts the real development. User interface design concentrates on

anticipating what actions users might perform and also making sure that the interface has elements that are easily understood and accessed. Designers have to choose the interface elements (i.e. Input controls, Navigational Components, Information Components, and Containers) that are consistent and predictable in layout which helps in-task completion with more efficiency and satisfaction [13].

3.2.2 User Interface Designers fundamentals or best practices

User interface designers must consider the following design fundamentals when starting on an interface [11][12].

1. *Know the user*

   "Obsess over customers: when given the choice between obsessing over competitors or customers, always obsess over customers. Start with customers and work backward." – Jeff Bezos.

   The user's goals should be the designer's goals. Hence a designer must learn them, restate them and repeat them. Also, a designer must know about the user's skills and experience. Focusing on the user's needs will let them achieve their goals.

2. *Keep the interface simple*

   "A modern paradox is that it's simpler to create complex interfaces because it's so complex to simplify them." – Pär Almqvist

   The best interface designs are almost invisible to the user. They do not contain unnecessary elements and are clear in the language used on labels and in messaging.

3. *Create consistency and use common UI elements*

   "The more users' expectations prove right, the more they will feel in control of the system and the more they will like it." – Jakob Nielson

Using common elements in UI makes users feel more comfortable and will cause them to get things done more quickly and efficiently. It is also important to create patterns in language, layout and design to help facilitate efficiency and consistency which helps users learn how to do something and transfer that skill to other parts of the application.

4. *Purposeful page layout*

   Considering the spatial relationships between items on the page, structure the page based on their importance. It helps draw attention to the most important pieces of information, increasing the readability.

5. *Strategically use of color and texture*

   "Designers can create normalcy out of chaos; they can clearly communicate ideas through the organizing and manipulating of words and pictures." – Jeffery Veen (The Art and Science of Web Design)

   Directing attention toward or redirecting attention away from items using color, light, contrast, and texture to your advantage.

6. *Typography to create hierarchy and clarity*

   Use of typeface to create different sizes, fonts, and arrangement of the text to increase scan ability, legibility and readability.

7. *Provide feedback*

   The interface should communicate what is happening i.e. inform users of location, actions, changes in state, or errors. The use of various UI elements to communicate status can help users know whether his or her actions have led to the expected result.

8. *Think about the defaults*

Thinking and anticipating the goals users bring to the site, create defaults that reduce the burden on the user. It is important when it comes to form design where there is an opportunity to have some fields pre-chosen or filled out.

3.2.3 The current trends and future of user interface

User interfaces are everywhere on websites, mobile phones, television, wrist watches, airplanes, washing machines etc. If there is a user, there is a user interface. Similarly, if there is a frustrated user, there is usually a case of bad user interface. The following picture depicts the evolution of user interface [13].

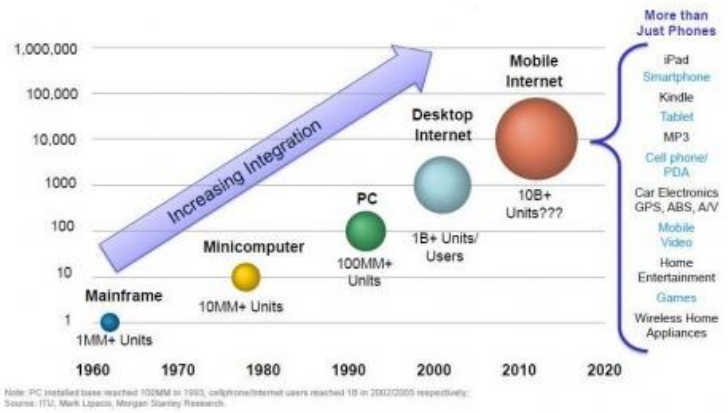


Figure 1: Evolution of the User Interface

3.2.3.1 The current trends of User interface

Minimalism: Going back to basics in user interface design, the glossy icons of websites windows 8 and Pinterest are replaced by simpler single color versions or text based buttons. It is a simple way to design web applications which mostly focuses on user generated content, so users usually might not complain about it. This might be a simpler technique which few users might not prefer, so check user preferences before using it [14].

Skeuomorphism: This user interface design was first popularized by Apple. This design approach relies on imitating the look and functionality of traditional and familiar real world objects to

make the interface more intuitive. For example, a wooden bookshelf with book covers to represent a digital library i.e. iBooks, this is generally used in designing mobile applications but not usually a best path to take for web applications [14].

Laser Focus: User interface has visual focus on a single, obvious task to do when a user opens the web application, instead of providing several equally important options. The key benefit of this approach is simplicity – users instantly know what the application is about and what the suggested action is. A good example of this is the Google homepage. This approach is useful in designing applications that have only a single, main function [14].

Context Sensitive Navigation: This approach came with the rise of dynamic user interface. Designers have to decide on which navigation elements should be on screen all the time and what can be shown only during certain actions. For example, Pinterest or Gmail usually shows the message action buttons only when the message is selected [14].

Collapsed Content: This approach is to hide non-essential options under one link which will expand and collapse on a user's request. For example, instead of showing a large number of links in the header, introducing a "More" button which will display a drop-down menu with links to pages that are not more important to the user [14].

Content Chunking: This approach makes reading and digesting large amounts of data easier, by presenting large amounts of data in smaller chunks. For example, splitting the article in smaller headings and adding images would make it easier and interesting to read [14].

Long Pages: Long pages require a lot of vertical scrolling assisted by a mouse wheel. Users are so accustomed to vertical scrolling that it's actually worse to split the content on separate pages which requires extra effort to find it and reach it [14].

3.2.3.2 Future of User interface:

Looking into future of user interface is worth the time, as it will change the way we work and the products we deliver**.** At this moment there is a large shift going on i.e. shift from desktop to mobile devices, and so this forces designers to refocus. One cannot know the future but can speculate [13].

The graphical user interface is not the future of the user interface, it is rather a current approach used to design an interface. It just completes the visual tasks to receive results such as clicking on a button to get a result etc.

The future enabling technologies are described as follows: [13]

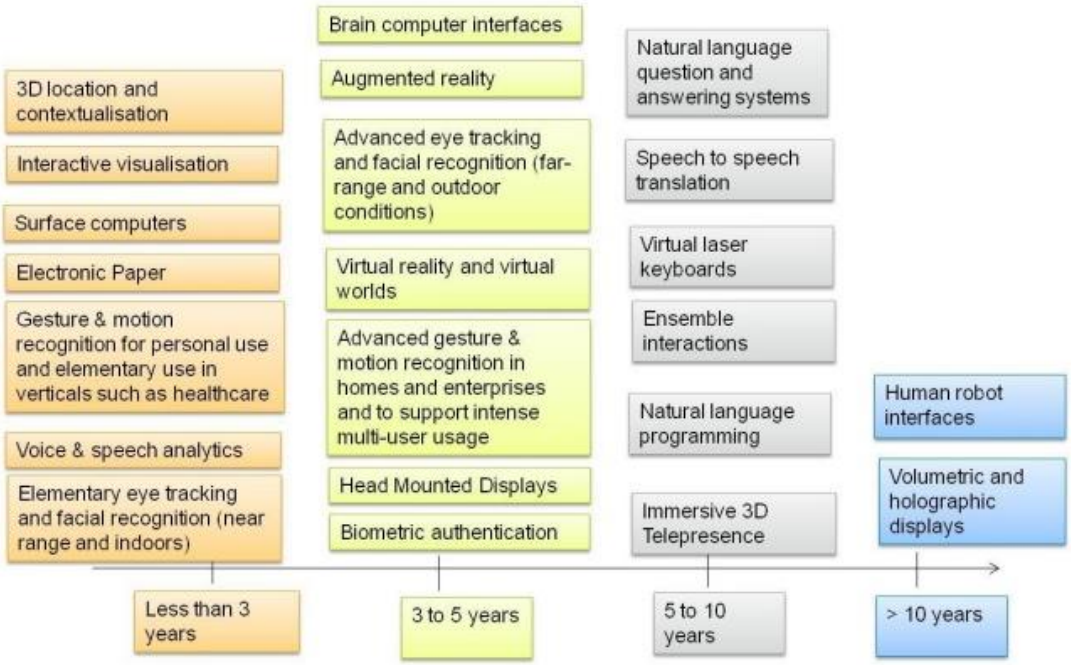


Figure 2: Timeline of enabling technologies

Electronic (e-) paper: Electronic or e-paper refers to reflective display technologies that do not require a backlight, which can be viewed in conditions of moderate to good ambient illumination. It can be made thin and produce a thin paper. The initial major applications for e-paper are small information- centric screens in mobile devices and music players.

Interactive Visualization: Displaying information/data using interactive images with colors, brightness, size, shape and motion of visual objects represents aspects of the dataset being analyzed. In security, visualization of cybersecurity data may uncover hidden patterns and identify emerging vulnerabilities and attacks, enabling a quick response with countermeasures.

Gesture Recognition and Motion Sensing: Determining the movement of a user's fingers, hands, arms, head and body in three dimensions through the use of camera or a device embedded with sensors that may be worn, held or bodily mounted.

Eye Tracking and Facial Recognition: Eye tracking technology facilitates determination of the angle or position of the user's visual attention using a camera. Generally non-intrusive eye trackers include two components: a light source and a camera. The light source is directed towards the eye while the camera tracks the reflection of the light source. The data obtained is used to extrapolate the rotation of eye and derive the direction of the gaze.

Facial recognition analyzes facial features and expressions of a user to infer information like age, gender and emotional state. For example, cameras placed on billboards and advertisements, profile the type of user in order to dynamically change the advertisement to suit the profile of the user.

Natural Language Question and Answering: This technology provides users with a means of asking questions in a plain language to computers/systems and receiving a response of high reliability and confidence in a reasonable time frame. A good example of this is IBM's Watson.

Virtual Laser Keyboards: Virtual laser projectors are small projector modules that display user interface (i.e. a keyboard or mouse) onto any flat surface so that users interact more easily with the mobile device or systems without the use of a physical keyboard or mouse.

Voice and Speech Analytics: Voice stress analytics is an elementary form of voice and speech analytics, which recognizes a person's stress level by analyzing the characteristics of a voice signal, with words used as an additional input. For example, it is used in call centers to identify an angry or abusive customer. Apple's Siri is also an example of speech analytics.

Speech to Speech Translation: Involves translation of one spoken language into another. It converges speech recognition, machine translation and text-to-speech capabilities.

Human-Robot Interfaces: The convergence of the technology's natural language question and answering system, speech to speech translation, gesture recognition, eye tracking and facial recognition may lead to the emergence of a truly intelligent human-robot interface that is able to navigate in various physical environments, perform complex tasks and communicate with humans.

3.3 Security

3.3.1 What is Security?

An application can have great features such as a firewall, passwords systems, etc. But if the rest of the system is not secured, attackers will drive right around the application's security. The previous generation of websites simply display data "Here's is where our store is, here's what we sell". But today, websites also take data from users: usernames, date of birth, email id etc. To retrieve this data, there needs to be a hole in the firewall. In other words, that hole is there intentionally, because we want to get data from our users. Developers tried to secure that hole in several ways. SSL (Secure Socket Layer) creates a secure pipe to the user, but if that user is malicious, SSL gives zero security. Automated systems such as intrusion detection, intrusion prevention, and app firewalls are great, and applications should absolutely use them. To that end, the only way to secure an application is to program the application to securely handle malicious data [9].

An example of insecure code being hacked with the malicious data [8]

*String SQLquery = "Select * from TABLE where name = '" + userdata1 + "' and pass = '"*
+ userdata2 + "'";

Expected Input: Bob, dogsname

Then query returns values of name "Bob" and password "dogsname"

*Select * from TABLE where name = 'Bob' and pass = 'dogsname'*

Attacker's input: Bob, foo' OR 'a' = 'a

Then query returns all the rows of name “Bob” as password value is always true.

*Select * from TABLE where name = ‘Bob’ and pass = ‘foo’ OR ‘a’ = ‘a’*

This is SQL injection [8]: we expected a password but instead got some SQL logic, which allows the attacker to log in without knowing Bob’s password. An easy fix to this without changing the code and performance either is to use a SQL prepared statement it actually runs faster and is more secure.

Fixed:

*PreparedStatement stmt = connection.prepareStatement (”Select * from TABLE where name = ? and pass = ?”);*

stmt.setString(1, userdata1);

stmt.setString(2, userdata2);

A prepared statement takes the form of a template into which certain constant values are substituted during each execution. Prepared statements are resilient against SQL injection, because parameter values are transmitted later using a different protocol.

3.3.2 Parts of Security and How Application Security fits in it

Business Practices are the rules, goals and procedures of the business. For example, “New employees get security training,” or “Only employees can access the internal network.” Business practices usually live in documents and peoples’ heads, and guide the implementation of code. Business practice security is mostly handled by managers and security professionals.

Network security is about preventing unauthorized network access. It uses firewalls and similar tools. Think of this as the implementation of the business practice, "Only employees can access the internal network." Network security is mostly handled by sys admins.

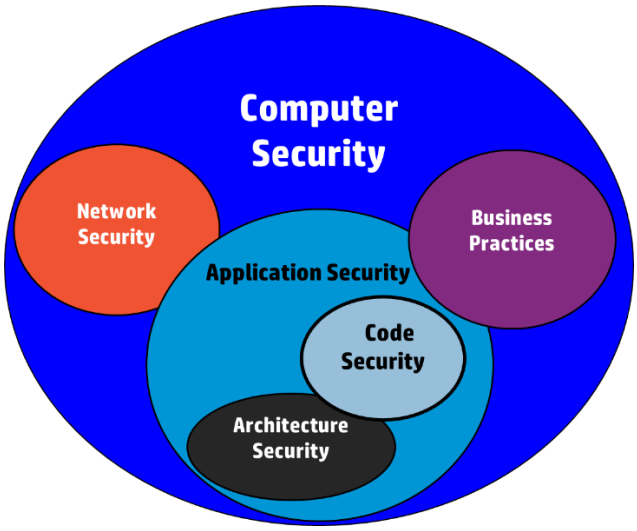


Figure 3: Parts of Computer Security

Application security ("app sec") is about how applications behave and how applications handle malicious data. Developers mostly handle application security.

There are several types of security within app sec. Two main ones are architecture security and code security.

Architecture security deals with how applications are put together. Is there a separation of presentation and data layers? How is session handled? Etc. Architecture security requires human experts to understand the architecture.

Code security [8] deals with how the application is implemented. When you take data from the user and send it to the database, did you validate that data? When you read in the user's

password, did you miss a null check that can cause the application to never properly authenticate the user? And so on. Application security scanners are generally built for code security. They can look through every line of code, hypothesize about millions of control flow paths, and generally do a much more thorough job than a human could.

3.3.3 Security Testing

Developers cannot build a secure application without performing security testing on it. It is a phase within the SDLC (Software Development Life Cycle). Security testing, by itself, is not a particularly good measure of how secure an application is because there are an infinite number of ways that an attacker can break into an application, and it is not possible to test of all them. However, security testing has a unique power to convince someone that there is a problem in an application. In order to make security testing effective, testers need to test early and test often, understand the scope of security, develop the right mindset, understand the subject, use the right tools, use source code when available, develop the metrics, and document the results [9].

3.3.4 Techniques for Security Testing

There are number of companies selling automated security analysis and testing tools. Limitations of these tools are they can be used for what they are good at. Most of the tools are generic i.e. they are not designed for application's custom code but for applications in general.

"Tools do not make software secure! They help scale the process and help enforce policy." – Michael Howard at 2006 OWASP AppSec Conference in Seattle [9].

There are four basic techniques for analyzing the security of a software application: [9]

1. Manual Inspections and Review
2. Threat Modeling

3. Code Review
4. Penetrating Testing

Manual Inspections and Review- Manual inspections are human-driven reviews that typically test the security implications of the people, policies, and processes, but can include inspection of technology decisions such as architecture designs. They are usually conducted by analyzing documentation or performing interviews with the designers or system owners. Not everything everyone tells you or shows you will be accurate.

Advantages:

- Requires no supporting technology
- Flexible
- Promotes Teamwork
- Early in SDLC
- Can be applied in variety of situations

Disadvantages:

- Can be time consuming
- Supporting material not always available
- Requires Significant human thought and skill to be effective

Threat Modeling- Threat modeling can be seen as a risk assessment for applications to help designers think about the security threats that their applications might face. In fact, it enables designers to develop mitigation strategies for potential vulnerabilities and helps focus their inevitably limited resources and attention on the parts of the system that most require it. Threat

models should be created as early as possible in the SDLC and should be revisited as the application evolves and development progresses. It has the same advantages and disadvantages as manual inspection and review.

Source Code Review- Secure Source Code review is the process of reviewing the source code of an application to verify that security controls are present, that they work as intended, and that they have been invoked in all the right places. It helps to identify any deviations from security requirements or security best practices as well as to identify security vulnerabilities in the source code. Secure code review is an activity of the development phase in the Secure Development Lifecycle (SDL).

Figure 4: Software Development Lifecycle

Techniques:

Manual: It is a bottom up approach performed without using a tool by text matching (e.g. using regular expressions), input/output path analysis, authorization logic validation, architecture analysis, etc.

Automated: Tool based automated code review techniques can assist in improving the throughput of the code review process. Static or dynamic source code analysis tools can analyze a program for hundreds of different security flaws at once (e.g. SQL injection, Cross Site Scripting etc.), at a rate far greater than any human can review code. Tools produce both false positive and false

negative results (e.g. a hardcoded password identified by the tool might not really be a password, such instances are called false positives).

Sample Tools: Fortify, Ounce, DevInspect, FxCop, PMD, Findbugs, Yasca, CodePro Analyticx(Security rule set), IBM Security AppScan Source

Approach and Methodology:

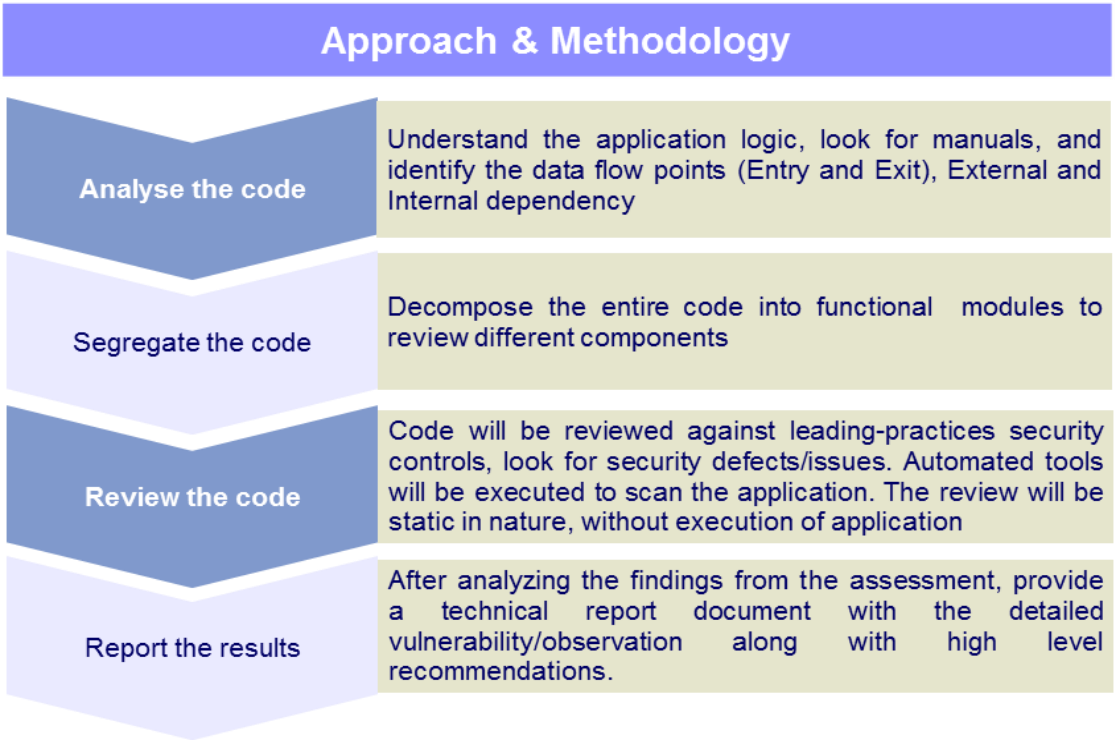


Figure 5: Secure Code Review Methodology

Advantages:

- Secure code review enables the development team to identify and correct insecure coding techniques that could lead to security vulnerabilities.
- *Effort benefit*: The effort to fix the vulnerabilities in the earlier stage of the SDLC process is much less than the later stage of the process. Last minute fixing may affect the entire functionality of the program and hamper deadlines set for product release.

- *Cost benefit*: Cost is directly proportional to effort required. Not only development cost, but also, a vulnerability identified in the production environment may involve more costs.
- *Compliance*: Some compliance, such as PCI, makes it necessary to do a secure code review before launching the product.
- *Reputation*: Secure code review removes most of the security flaws in the earlier phase, making it more secure than just doing black box assessments. So there is less chance of the product being compromised, hence lesser chance of reputation damage.

Penetration Testing: It has been a primary security testing technique for testing network security for many years. It is also commonly known as black box testing or ethical hacking. It is essentially the "art" of testing a running application remotely, without knowing the inner workings of the application itself, to find security vulnerabilities. Typically, the penetration testing team would have access to applications as if they were users. The testers are given a valid account on the system, act like attackers and try to find vulnerabilities. Penetration testing tools have been developed that automate the process.

Advantages:

- Can be fast and cheap
- Requires a relatively lower skill-set than source code review
- Tests the code that is actually being exposed

Disadvantages:

- Too late in SDLC and front impact testing only

3.3.5 Security Risks and Protection Measures in online learning

Security is essential as a means to retain users' trust in the online education environment because any risk can dramatically affect students' perception of a system's reliability and trustworthiness [5]. Online education faces various security risks, which mainly comes from external intruders. Below mentioned are some of the security risks in online education [5].

- Brute Force Attack
- Cross-Site Request Forgery (CSRF)
- Cross Site Scripting (XSS)
- Denial of Service (DoS)
- IP Spoofing
- SQL Injection
- Session Hijacking
- Session prediction
- Rootkits
- Cache poisoning

As a result, it is crucial to identify the counter measures to mitigate these risks. Researchers have offered quite a few protection proposals [5]. Some of them are mentioned below:

- Installing firewalls and anti-virus software
- Implement Information Security Management (ISM) to build an effective security architecture that can fight existing and emerging information security threats.
- Improving authentication, authorization, confidentiality, and accountability
- Using digital right management
- Training security professionals

Security policies and mechanisms in online education must support authentication, authorization, confidentiality, and accountability. Authentication refers to the validation of a person's identity before the access is assigned. Authorization defines what rights and services a person can access after the authentication process is passed. Confidentiality means that some specific information or data cannot be disclosed to anyone who is not authorized. Accountability refers to the methodology by which users' resource consumption information is collected for billing, auditing, and capacity-planning purposes [5].

3.3.6 Qualifying and Quantifying the System Security Improvement

The focus of threat protection is moving from securing the infrastructure to securing the software applications that businesses write and deploy. The shift has created a market for a new generation of products and services known as Software Security Assurance (SSA) solutions that help companies uncover vulnerabilities in their code, effectively fix these defects, and produce software that is impervious to security threats [7].

In an effort to quantify the business value of SSA, Mainstay Partners studied 17 organizations that have implemented solutions from Fortify Software, a leading provider of SSA solutions. The study combined executive interviews, industry research, and benchmark analysis to identify, qualify, and quantify the full range of benefits that organizations are seeing from their SSA investments. The study found that companies are realizing substantial benefits from SSA right out of the box, saving as much as $2.4M per year from a range of efficiency and productivity improvements, including faster, less-costly code scanning and vulnerability remediation and streamlined compliance and penetration testing [7].

Application security meant protecting the software that is running in all these environments and devices, and the business improvements of SSA were seen as extending to wherever applications were deployed. The study reported a number of significant operational and financial improvements from the SSA implementations. A selection of key performance improvements are shown in the table below:

Performance Metric	Improvement
Vulnerabilities per application	From 100s to 10s
Average time to fix a vulnerability	From 1 to 2 weeks to 1 to 2 hours
Percentage of repeat vulnerabilities	From 80% to 0%
Compliance and penetration testing effort	From ~$500k to ~$250k
Time-to-market delays due to vulnerabilities	From 4+ incidents (30 days each) per year to none

Figure 6: Software Security Metrics and Improvement

By analyzing such improvements, identified are the following benefit areas for SSA-enabled organizations: [7]

- More efficient and effective vulnerability assessment and remediation
- Streamlined regulatory compliance and penetration testing efforts
- Fewer security-related delays affecting the launch of new products
- More favorable pricing of outsourced code development

Key Findings

Faster Vulnerability Remediation: Companies adopting SSA solutions reported significant efficiency improvements in finding and remediating software security flaws. Earlier in pre-SSA environment, vulnerabilities took an average of 1 to 2 weeks to fix. By introducing automated SSA technology and best practices, organizations reduced average remediation from 1 to 2 weeks to 1 to 2 hours [7].

"After implementing Fortify, a financial company uncovered thousands of previously unknown security flaws in its applications. By cleaning code early, the company is now avoiding remediation costs of around $1M per year, eliminating 100 hours of compliance testing per application, and avoiding product launch delay- a benefit worth $7M–$8M annually."

Streamlined Compliance and Penetration Testing: Most of the companies surveyed are facing tighter government and industry regulations for application security, particularly in new software standards in the financial services and health-care industries. SSA solutions helps control costs by streamlining regulatory compliance projects that require meeting strict application security standards. For example, organizations quickly identified and ranked vulnerabilities according to severity. The solution also generates a report that documents these activities, creating an audit trail for regulatory.

Avoiding Software Compliance Penalties: Businesses that fail to comply with industry standards for software security can face substantial penalties. In the payment card industry, for example, penalties can range from $5K to $25K per month.

"*Deploying Fortify initially as a proof-of-concept in a small department, a public-sector organization saw adoption spread quickly when security scans of mission-critical software uncovered 100 times more vulnerabilities than were known before."*

Avoiding Data Breaches: The threat of a major data breach can keep CISOs awake at night, and most are aware of the history of high-profile security failures that have damaged company reputations and resulted in millions of dollars in legal, remediation expenses, lost revenue, and customer churn [9].

4. System Framework

4.1 FYFL System Architecture

4.1.1 2-Tier Architecture

The FYFL system has 2-tier architecture. It consists of a web browser and a web server in one tier and a Database server in the second tier. It has a web browser as front end that sends requests to the web server and the webserver retrieves data from the database and sends requests to the database server.

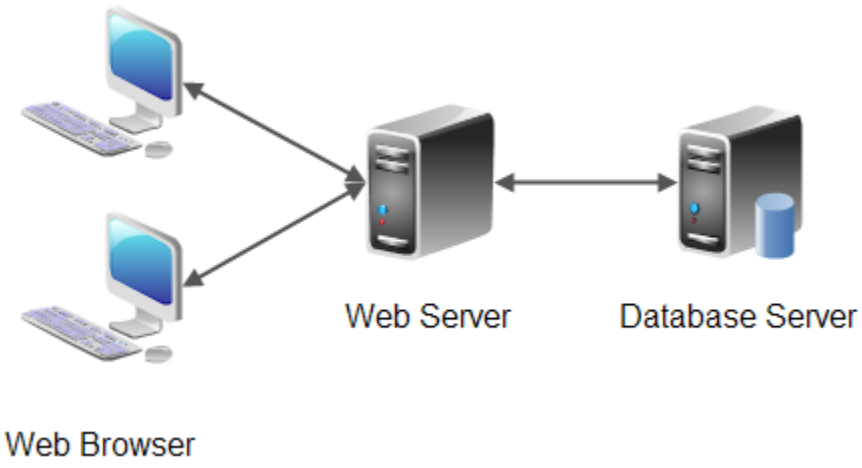


Figure 7: 2-Tier Architecture

4.1.2 Web Application Architecture

The figure below depicts the flow of the web request through the browser, the Internet, the web server, the application server, and the database server.

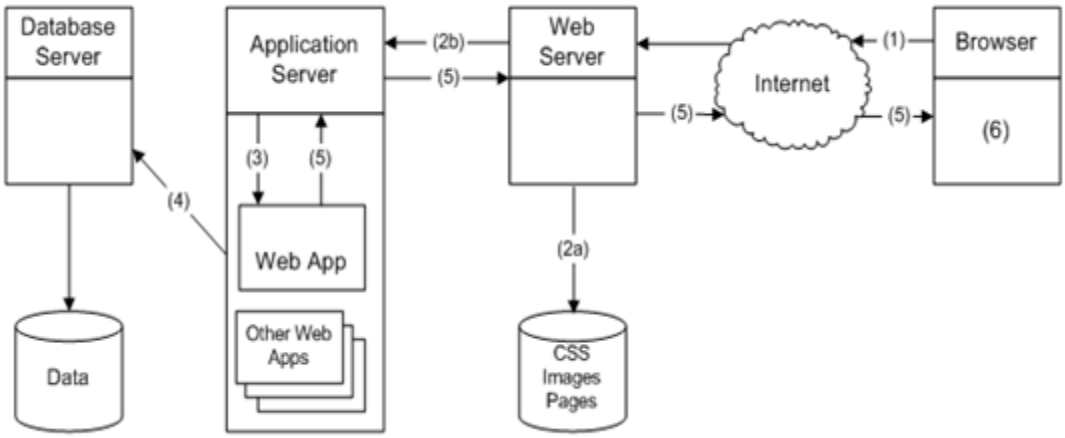


Figure 8: Web Application Architecture

4.1.3 Web Application Security Architecture

The figure below represents methods to be implemented at different stages of architecture to improve the security of the web application.

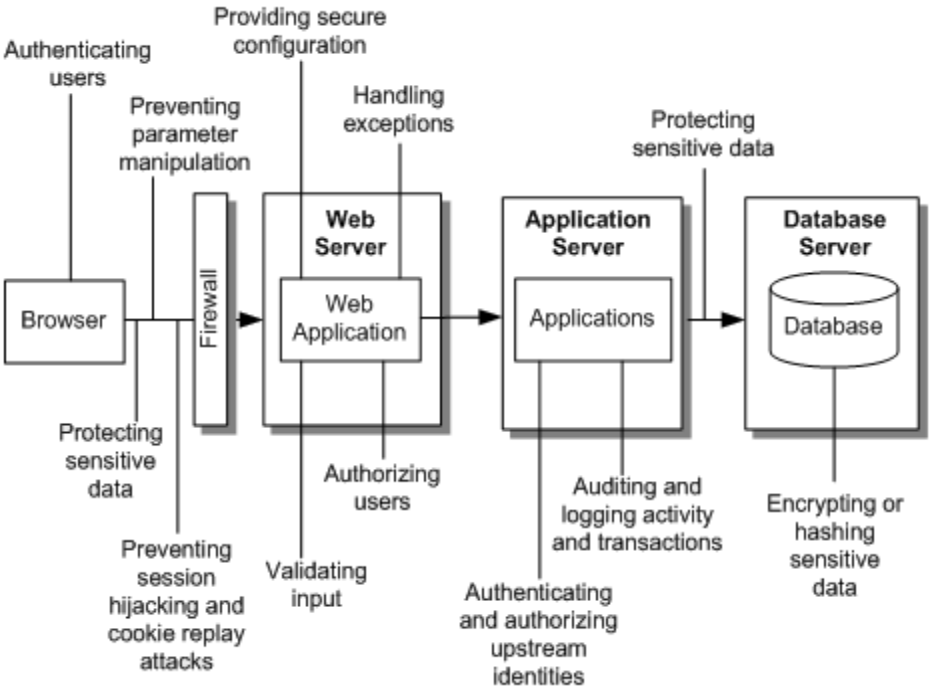


Figure 9: Web Application Security Architecture

4.2 FYFL System Use Case Diagram

Use case diagram depicts the actors, the functionalities represented as a use case and relationships among the use cases and actors. The figure below represents the use case diagrams of the FYFL system with the user and the admin as actors and login, registration, create a badge, search a badge, approve a badge, add a quiz, manage a user, create a group/club, earn a badge etc. as functionalities.

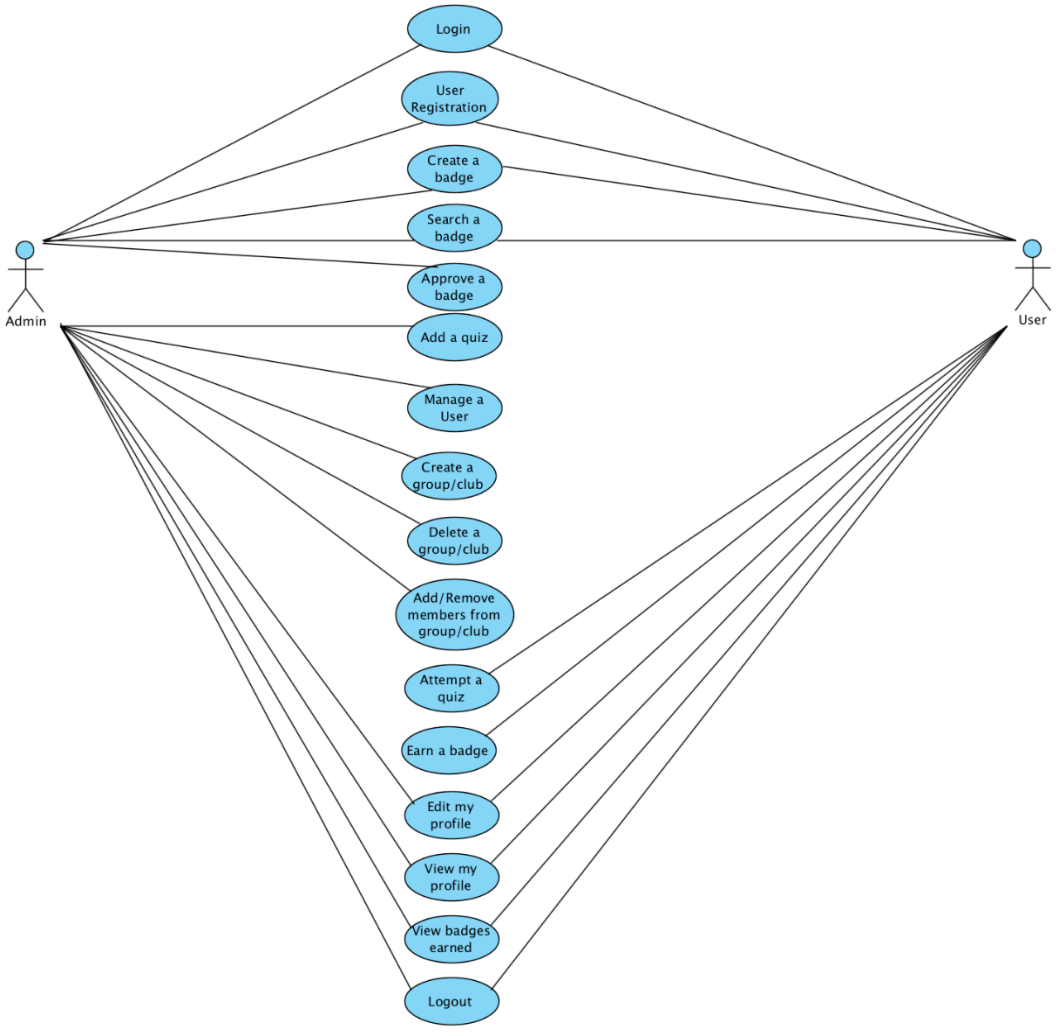


Figure 10: Use Case diagram

4.3 FYFL System Activity diagrams

An activity diagram is another important diagram in UML to describe a dynamic aspect of the system. An activity diagram is basically a flow chart to represent the flow from one activity to another activity. The flow can be branched or concurrent and describes the sequence from one activity to another. Below are a few activity diagrams of the FYFL system.

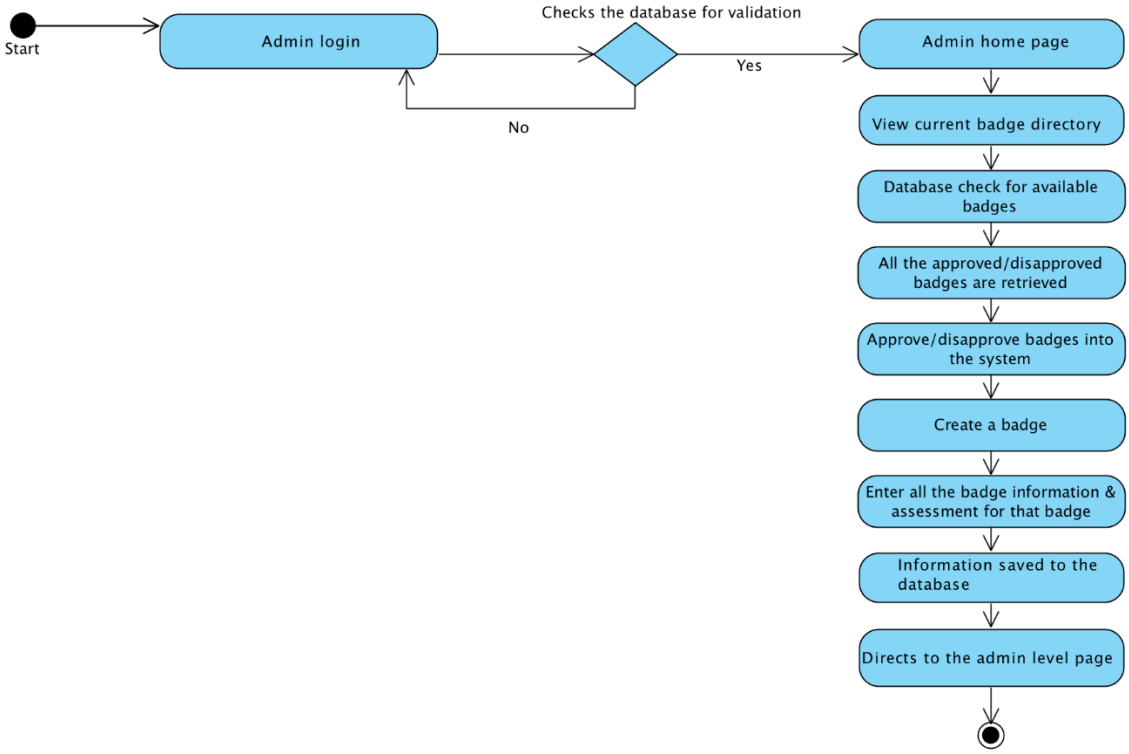


Figure 11: Admin functionality - Approve a badge and create a badge

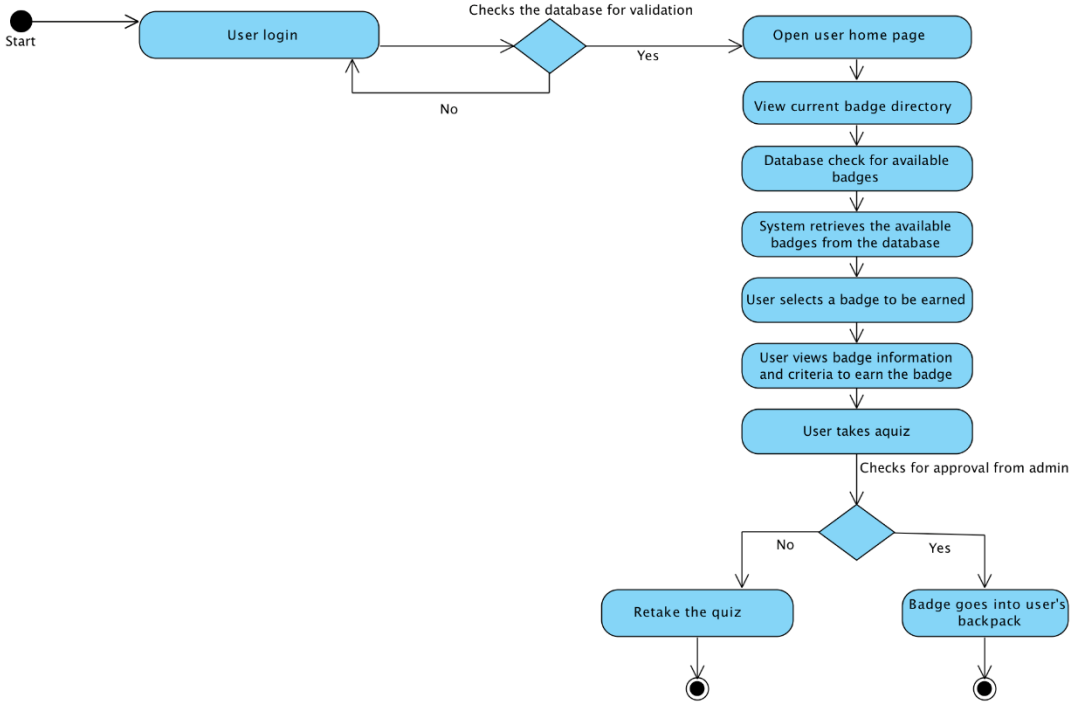


Figure 12: User Functionality - Earn a badge

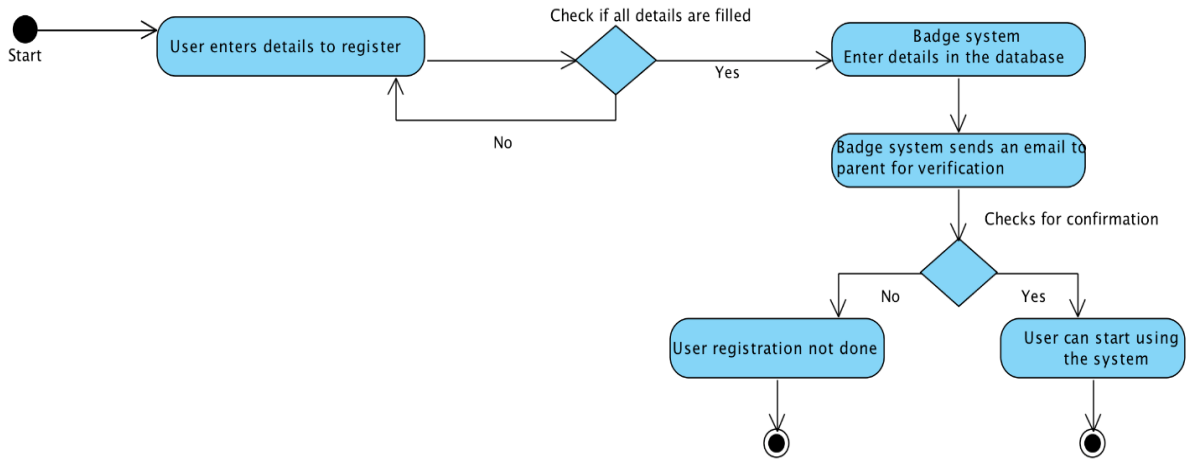


Figure 13: User Functionality – Registration

4.4 Database Architecture

The current database architecture of the entire FYFL system is depicted below. The database consists of tables Hands_Response, Mechatronics_Response, new_Images, users, back_packs, Competition_Response, badges, groups, states, admin1, counties etc. related as shown below. For example, a table Mechatronics_Response containing badge name, badge id, username, user id is related to a table images3 containing badge name and respective badge image information i.e. image name, type, size, date etc.

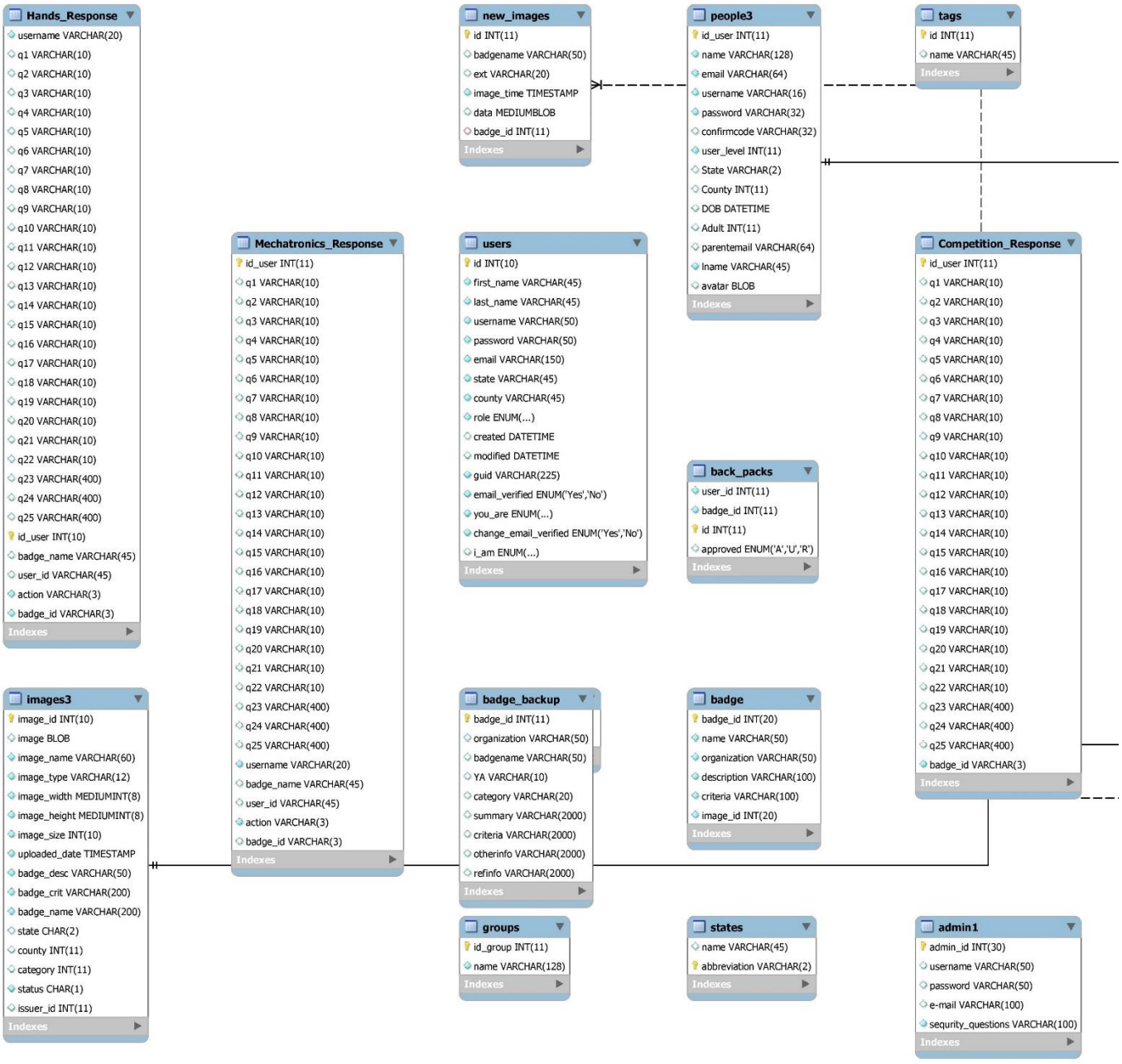


Figure 14: Entire FYFL system database architecture

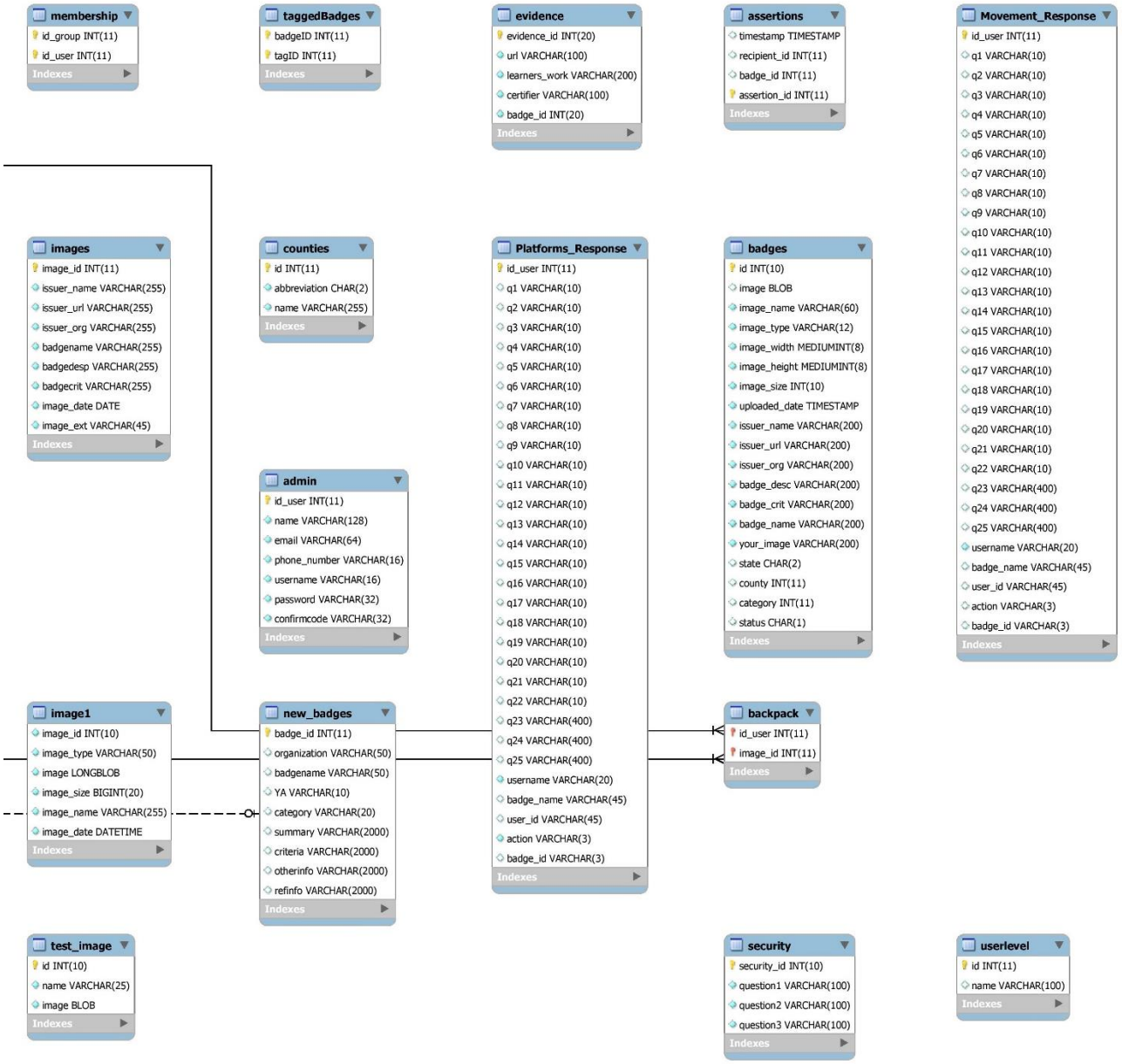


Figure 15: Continuation of Database Architecture

The figure below depicts database's tables that can be accessed from the files identified for having security vulnerabilities. Mitigating the SQL injections protects these tables from unauthorized access.

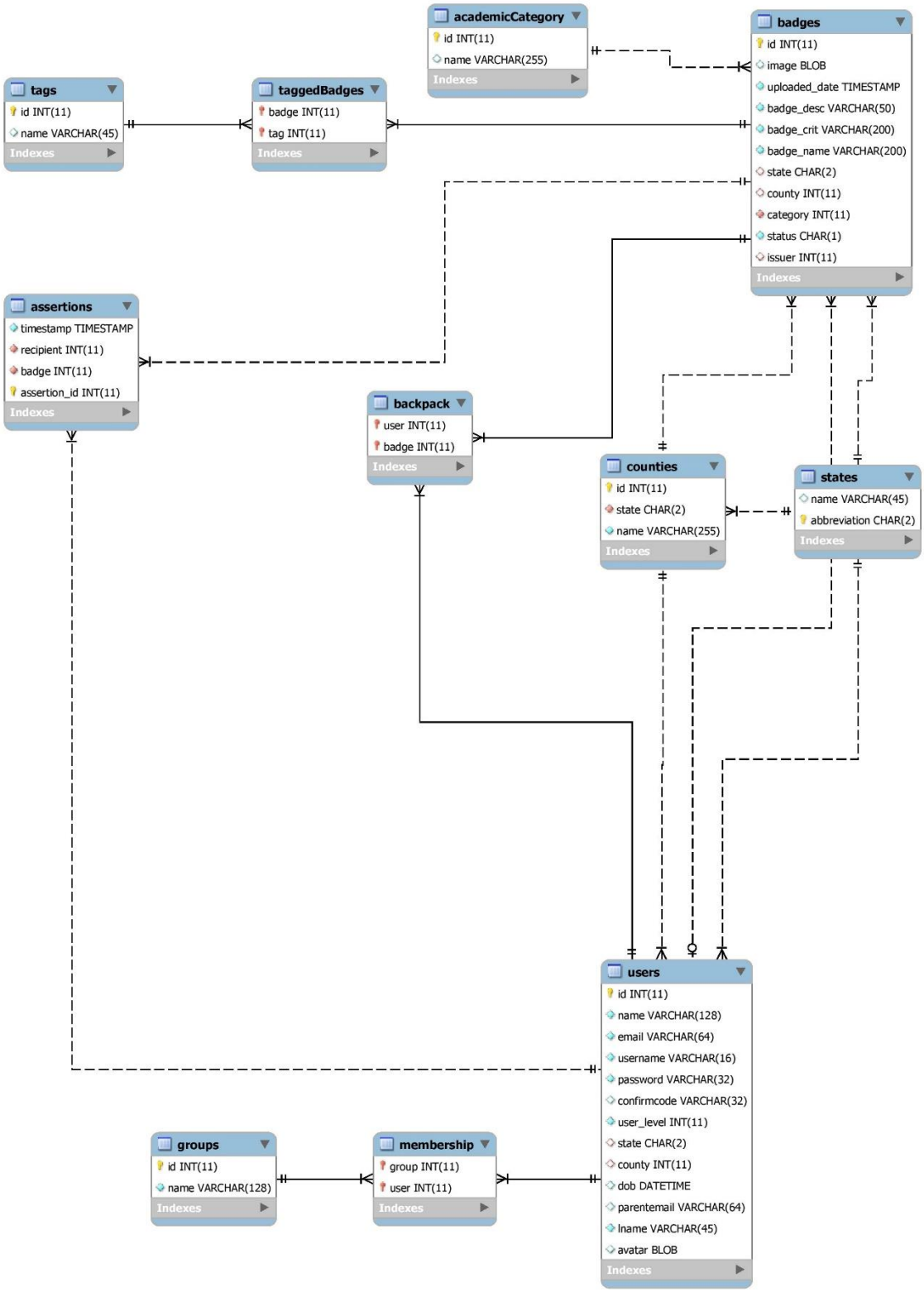


Figure 16: Protected tables

5. Security Testing Tools

5.1 HP Fortify

HP Fortify is a commercial security tool that helps reduce development cost by identifying vulnerabilities early in the SDLC. Also, it brings development and security teams together to find and fix security issues. It has three major components: Fortify Static Code Analyzer, Software Security Center, and Audit Workbench.

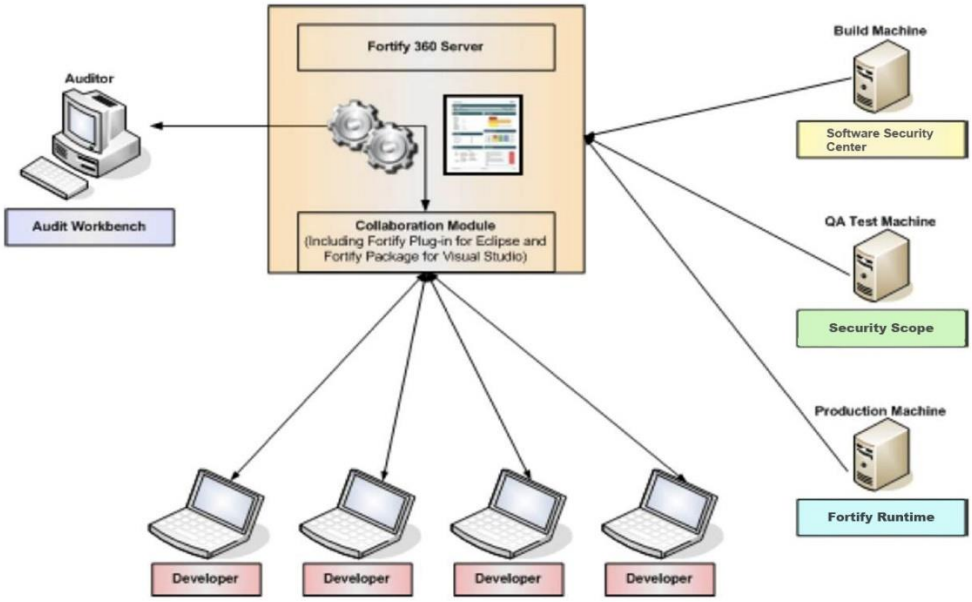


Figure 17: Components of HP Fortify

The middle box is the Fortify Server product, software security center. It's a WAR (Web Application Archive), and runs in Tomcat or other app servers. Everyone who works with Fortify should have an account in Software Security Center (SSC). SSC will collect results from Source Code Analyzer (SCA) scans, Security Scope and Fortify Runtime.

Each developer machine gets Fortify SCA (to run local scans) and either Audit Workbench or the IDE plugin (to view scans and fix issues). After auditing a scan, upload that scan into SSC. The build server (where code is built) should also get Fortify SCA, so it can run scans automatically. It will upload those scans to SSC, so they are available for the team. The source code can be scanned either from the IDE plugin (such as Microsoft Visual Studio, eclipse) or Audit Workbench.

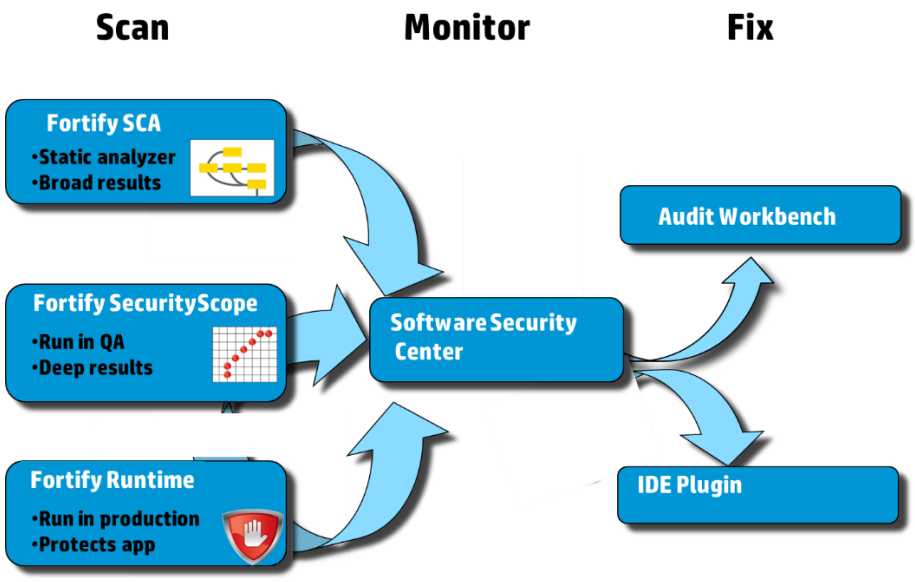


Figure 18: Process flow

HP Fortify Static Code Analyzer (SCA): Provides automatic static code analysis to help developers eliminate vulnerabilities and build secure software. It also helps verify that the software is trustworthy, reduces costs, increases productivity and implements secure coding best practice. The static code analyzer scans source code, identifies root causes of software security vulnerabilities and correlates and prioritizes results, giving line of code (LOC) guidance for closing gaps in the security. To verify that the most serious issues are addressed first, it correlates and prioritizes results to deliver an accurate, risk ranked list of issues.

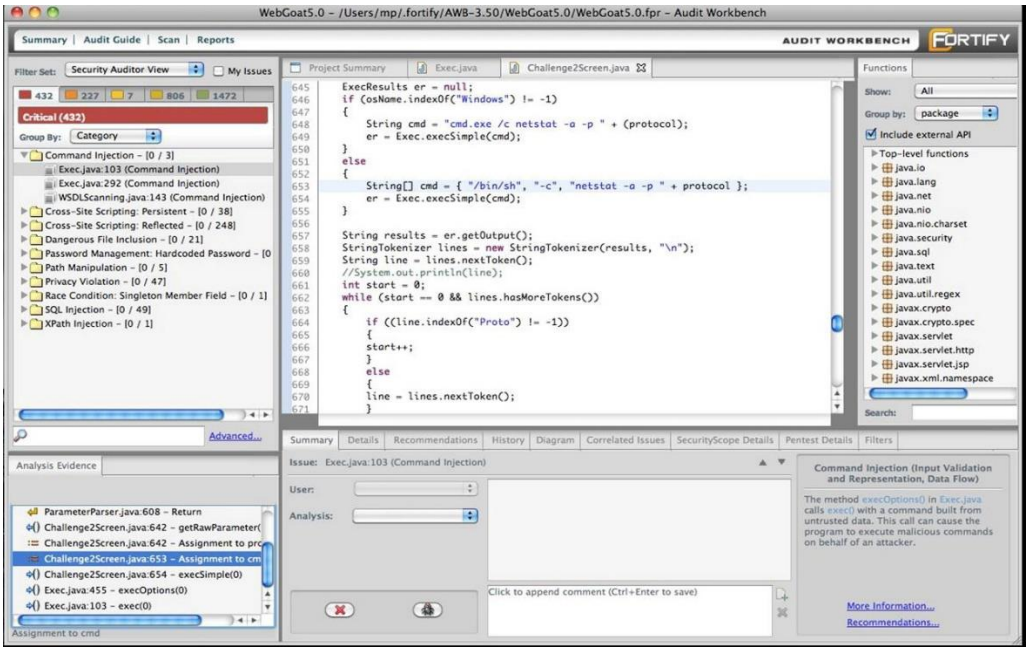


Figure 19: HP Fortify Source Code Analyzer

HP Fortify Software Security Center: The scanners can put data into the Software Security Center, which you can think of as version control and reporting for security scans. Users can view data in the Software Security Center to fix code. Users or scanners can download the scan and open it in either IDE (using the Fortify plugin) or Audit Workbench (Fortify's IDE).

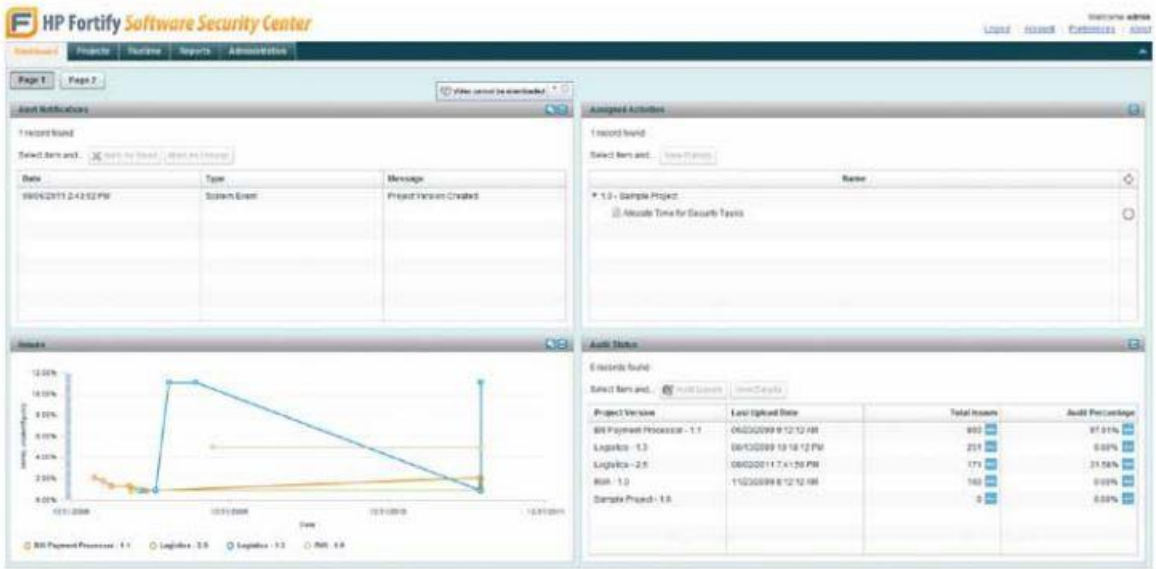


Figure 20: Software Security Center Dashboard

HP Fortify Software Security Center provides the ability to eliminate risk in existing applications and deliver new applications with security built in.

HP Fortify Audit Workbench: Audit workbench (AWB) can be used to scan the source code. In the Audit workbench front screen, select "Advanced Scan" then select the directory to scan i.e. the project's root.

Figure 21: HP Fortify Audit Workbench

AWB has five panels. After running a scan, the upper middle panel displays the initial project summary, the upper left panel lists the vulnerabilities sorted by severity (Critical, High, Medium, Low), the upper right panel lists the packages involved in the project, the lowest left panel is the analysis evidence, the lowest right panel has a summary of the issue, details about the issue, and recommendations about how to fix an issue. When an issue is clicked on, it will show the code related to that issue. Analysis evidence is a stack trace of the vulnerability, showing each function and each step that occurs as the vulnerability is exploited. Clicking on a line in this panel will jump to that line in the upper-middle code review. AWB helps in finding issues of a certain category, Auditing and suppressing issues, grouping issues, making a recommendation and fixing issues.

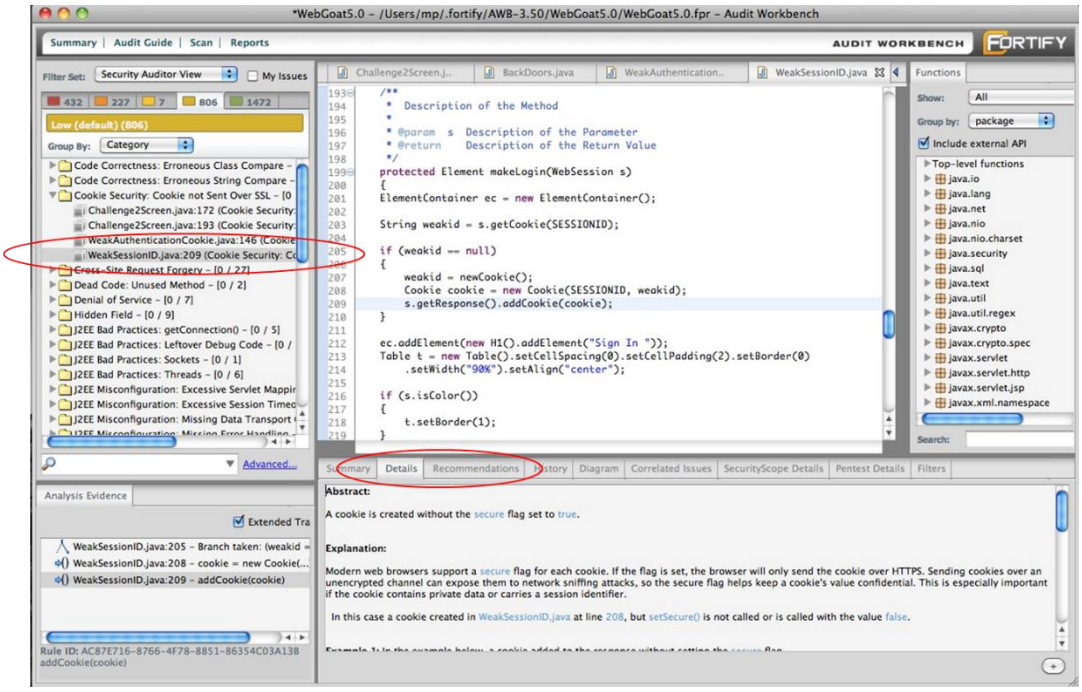


Figure 22: Audit Workbench - Making a Recommendation

5.2 Fortify on Demand

A managed application security testing service that enables organizations to quickly test the application security of a few applications or launch a comprehensive security program without additional investment in software and personnel.

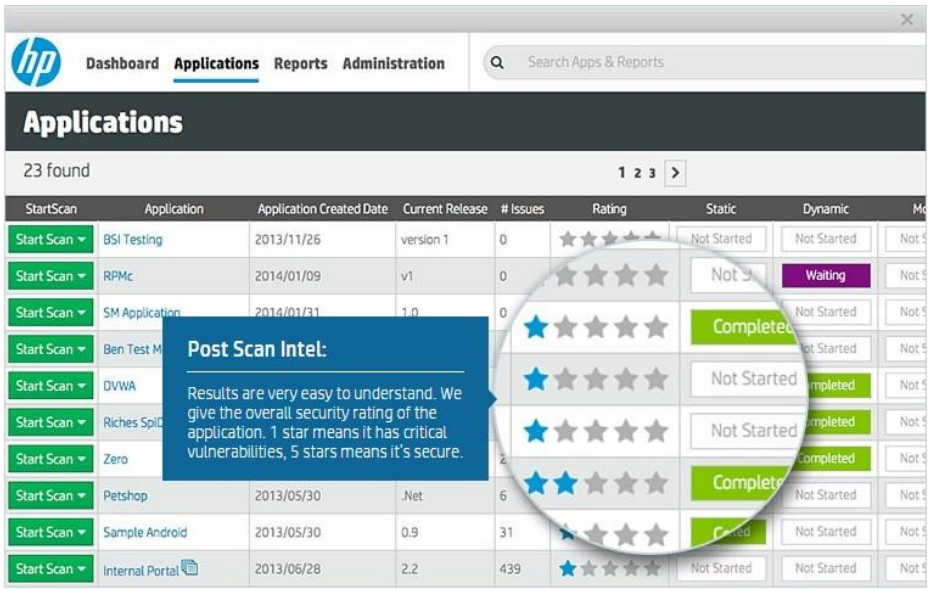


Figure 23: HP Fortify On Demand

Key Benefits:

- Managed security services: Without the need to hire a security testing team or install expensive hardware or software
- Quickest time to results: On an average 1 day for static and 3-5 days for dynamic
- Global Data Centers: Facility to host instances at global data centers
- Centralized Portal: Manage application security programs, coordinate testing schedule, manage remediation projects and collaborate across teams through one centralized interface. Also, generate reports and export results to upload into other reporting systems
- Comprehensive security testing: Across static, dynamic, mobile, vendor, and open source applications

Services:

- Static Testing
- Dynamic Testing
- Mobile Application Security Testing
- Digital Discovery
- Vendor Security Management

5.3 IBM Security AppScan Source

IBM Security AppScan Source is a commercial tool similar to HP fortify that helps organizations lower costs and reduce risk exposure by identifying and fixing web-based and mobile application source code vulnerabilities early in the software development lifecycle, which includes support for JavaScript, HTML5, Cordova, Java and Objective-C. AppScan components comprise the following service offerings:

IBM Security AppScan Standard is a desktop software for an automated application security vulnerability testing environment for IT security auditors and penetration testers and helps decrease the risk of web application attacks and data breaches. It includes glass-box testing with runtime analysis to identify more vulnerabilities, simplify scan configurations and provide more actionable results.

Figure 24: AppScan Standard glass-box testing

IBM Security AppScan Source is a software that prevents data breaches by locating security flaws in the source code. It provides assessment summaries that map to application risk and provide insight into vulnerabilities that affect your applications.

IBM Security AppScan Enterprise enables organizations to mitigate application security risks and achieve regulatory compliance. It helps security and development teams collaborate, establish policies, scale testing, and prioritize and remediate vulnerabilities throughout the application lifecycle. It delivers more than 40 compliance reports that map specific regulatory requirements to identified vulnerabilities, while tracking issue resolution.

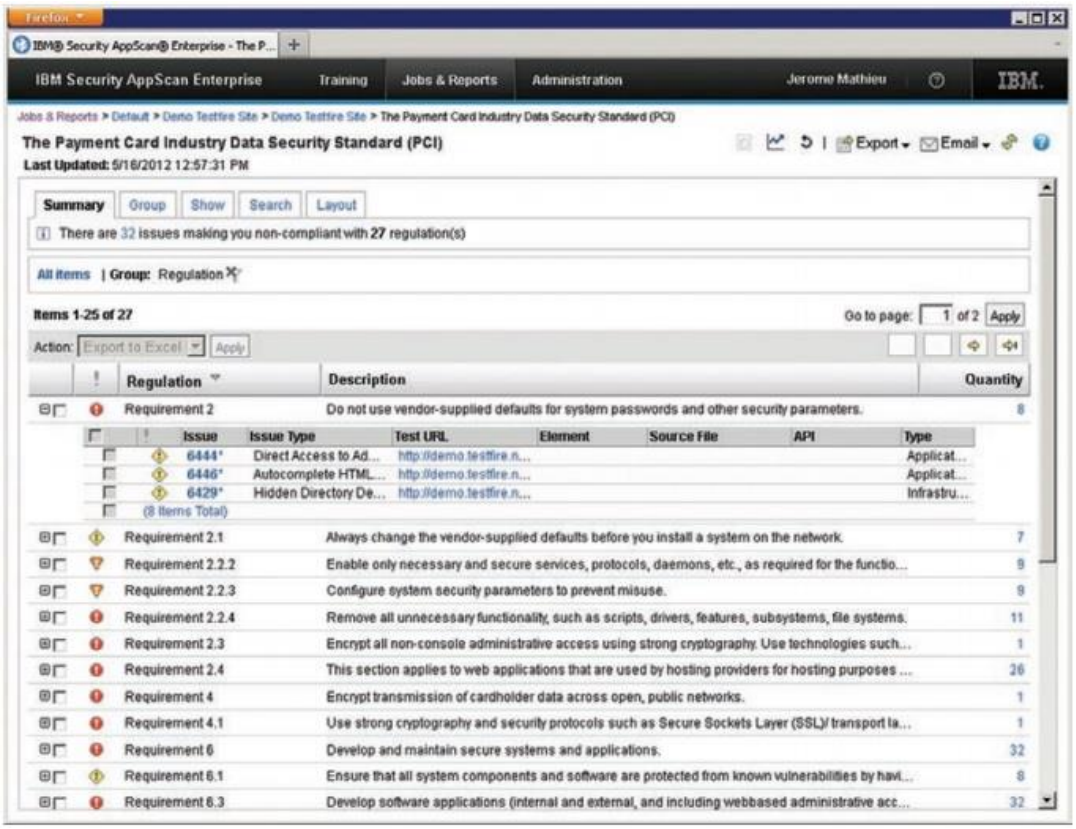


Figure 25: AppScan Enterprise Software

IBM Security AppScan Reporting Console can be used as an add-on for reporting.

Features:

IBM Security AppScan Source can enable:

- Stronger and cost-effective software security through source code analysis
- Improved intelligence through integration with existing tools and processes such as application development, build integration and security monitoring
- Security best practices through centralized management and enforcement of security policies
- Reporting, governance and compliance capabilities that facilitate communication of security status and issues

5.4 FindBugs

FindBugs is an open source program to find quality and security bugs in Java programs. It usually performs static analysis and looks for instances of "bug patterns" i.e. code instances that are likely to be errors. The tool is platform independent and needs a runtime environment compatible with Java 2 Standard Edition. Errors are classified into four ranks: Scariest, Scary, Troubling, and of Concern.

Listed below are few standard quality and security bug patterns reported by FindBugs version 3.0.0

1. *Method invokes System.exit( )* - Invoking System.exit shuts down the entire Java Virtual machine. It should be invoked when it is appropriate. Such calls make it hard or impossible for the code to be invoked by other code. Consider throwing Runtime Exception instead. It is a bad practice.
2. *Comparing of String parameter using "==" or "!="* - This code compares a java.lang.String parameter for reference equality using the == or != operators. This code requires callers to pass only String constants or interned strings to a method is unnecessarily fragile and may also lead to measurable performance gains. Consider using the equals(object) method instead. This is a bad practice.
3. *Format string should use %n rather than \n* - This format string includes a newline character (\n). In format strings, it is generally better to use %n, which will produce the platform-specific line separator. It is a bad practice.
4. *Iterator next() method can't throw NoSuchElementException* - This class implements the java.util.Iterator interface and its next() method is not capable of throwing java.util.NoSuchElementException. Therefore, the next() method should be changed so it

throws NoSuchElementException if it is called when there are no more elements to return. It is a bad practice.

5. *Field should be both final and package protected* - A mutable static field could be changed by malicious code or by accident from another package. The field could be made package protected and/or made final to avoid this vulnerability. This is a malicious code vulnerability.
6. *Hardcoded constant database password* - Code that creates a database connects using a hardcoded, constant password. Anyone with access to either the source code or the compiled code can easily learn the password. It is a security issue.
7. *Empty database password* – Code that creates a database connects using a blank or empty password. This indicates that the database is not protected by a password. It is a security issue.
8. *HTTP cookie formed from untrusted input* – Code that creates a HTTP cookie using an untrusted HTTP parameter, and when this cookie is added to a HTTP response header sent to a web user without being validated for malicious characters, leads to a HTTP response splitting vulnerability. It is a security issue.
9. *Non-constant string passed to execute method on an SQL statement* - The method invokes the execute 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. It is a security issue.
10. *JSP reflected cross site scripting vulnerability* - The code that directly writes HTTP parameter to JSP output, without proper input validation for malicious content allows reflected cross site scripting vulnerability. It is security issue.

5.5 Differences between commercial tools HP Fortify and IBM AppScan

Fortify SCA is a static code analysis tool	AppScan is a dynamic application testing tool
Performs white box testing to analyze vulnerabilities in source code	Performs black box testing to identify vulnerabilities in the application
Fortify SCA needs the entire source and libraries to compile the code	AppScan needs the application URL for testing as ready to test in a validation-testing environment

6. Research Plan

6.1 Proposed Hypothesis

As stated in section 2.4, the system should be secured against possible attacks like SQL injection, Cross Site Scripting, Cross Site Request Forgery, etc. Thereby increasing the learners' trust when using the securely improved site compared to the original site.

6.2 Preliminary Design

The study identifies three stages that the system go through on the path of Software Security Assurance maturity. There are Explore, Accelerate and Optimize.

Explore: We deploy a Software Security Assurance solution "HP Fortify" on the system and developer team as a proof-of-concept initiative.

Accelerate: Actively incorporating threat detection and remediation techniques across key development teams and the system.

Optimize: Embed Software security tools, processes, and training within a formal SDLC program. Also, leverage SSA solutions in innovative ways to generate additional business value and create competitive differentiation among the organization's competitors.

Figure 26: Path of Software Security Assurance maturity

6.2.1 Technologies used to build "For Youth For Life" System

HTML5 (Hyper Text Markup Language)

HTML5 is a core technology markup language of the Internet, which gives developers more flexibility, enabling them to develop exciting, interactive, powerful and efficient web applications that provide a vibrant user experience.

JavaScript

JavaScript is a dynamic computer programming language along with HTML5 that can provide a rich front-end with the fastest experience to web application users.

CSS (Cascading Style Sheet)

Cascading Style sheet is a style sheet language that specifies style and design formatting for websites such as color, font, size, and layout rather than content. It is most commonly used on HTML pages.

PHP

PHP is a server side scripting language for web application development and also used as a programming language. It can be used with HTML or any other templating engines and web frameworks. It is usually processed by a PHP interpreter.

6.3 Preliminary Experiment

6.3.1 Fortify Scan of the initial code

Considered the entire source code of the FYFL system, performed the static analysis using Fortify on Demand. Fortify on Demand SCA Scanner has scanned all the files of the source code and has identified the risks involved by categorizing them into severity levels. In total, there are 544 issues identified of which the severity of 106 issues is critical, 409 issues is high, and 20 issues low. The most prevalent issues identified are:

- **SQL Injection**: Invoking a SQL query built using input coming from an untrusted source. This could allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands. Constructing a dynamic SQL statement with input coming from an untrusted source might allow an attacker to modify the statement's meaning or to execute arbitrary SQL commands.
- **Privacy Violation – Autocomplete:** With auto completion enabled, some browsers retain user input across sessions, which could allow someone using the computer after the initial user to see information previously submitted.
- **Cross-Site Request Forgery:** The HTTP request must contain a user specific secret in order to prevent an attacker from making unauthorized requests. A Cross-Site Request Forgery vulnerability occurs when a web application uses session cookies, the application acts on HTTP requests without verifying that the request was made with the user's consent.

- **Cross-Site Scripting-Reflected:** Sending un-validated data to web browser, which can result in the browser executing malicious code. In case of Reflected XSS, the untrusted source is typically a web request.
- **Cross-Site Scripting-DOM:** Sending un-validated data to web browser, which can result in the browser executing malicious code. In the case of DOM-based XSS, data is read from a URL parameter or other value within the browser and written back into the page with client side code.
- **Password Management-Hardcoded Password and Empty Password:** Hardcoded passwords and empty passwords could compromise system security in a way that cannot be easily remedied.
- **Password Management-Insecure:** Password as part of an HTTP GET request will cause the password to be displayed, logged and stored in the browser cache. The parameters associated with an HTTP GET request are not treated sensitive data.

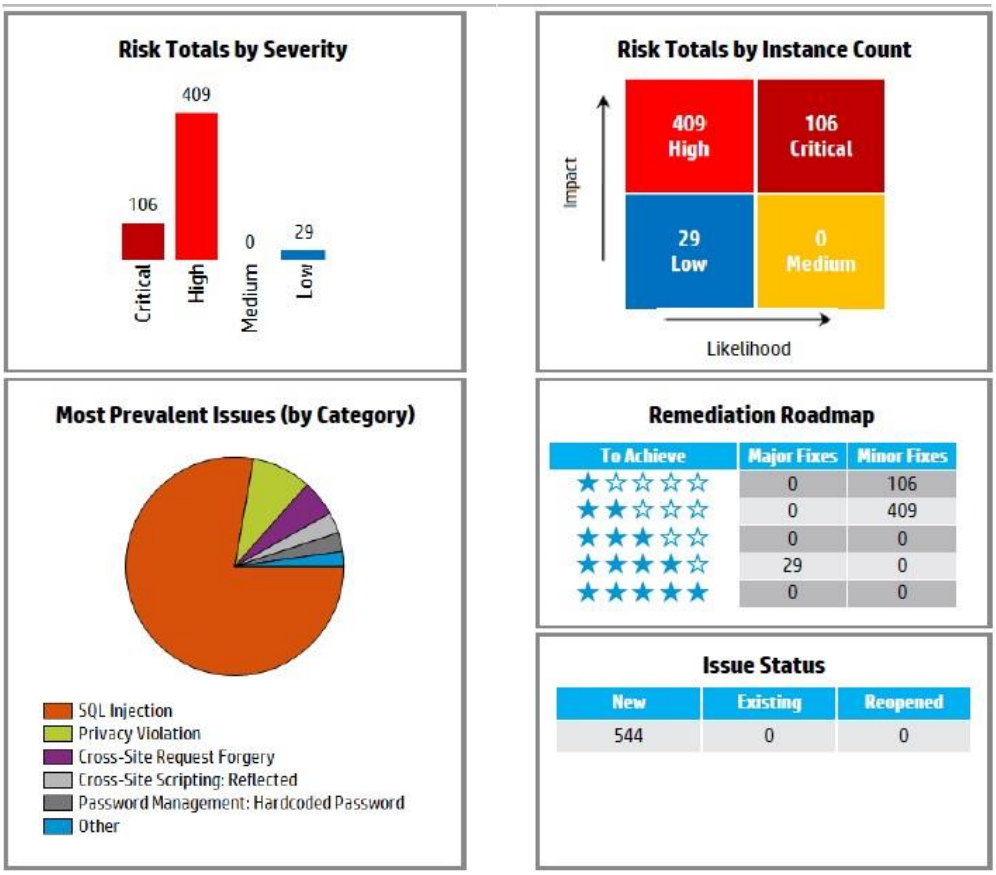


Figure 27: Executive Summary

Likelihood is the probability that a vulnerability will be accurately identified and successfully exploited. Impact is the potential damage an attacker could do to assets by successfully exploiting a vulnerability. This damage can be in the form of, but not limited to, financial loss, compliance violation, loss of brand reputation, and negative publicity. Critical-priority issues have high impact and high likelihood. As such should be remediated immediately. SQL injection for an example is an example of a critical issue. High priority issues have high impact and low likelihood. Medium- priority issues have low impact and high likelihood. Low-priority issue have low impact and low likelihood.

Issue Breakdown

Issues are divided based on their impact (potential damage) and likelihood (probability of identification and exploit).

High impact / high likelihood issues represent the highest priority and present the greatest threat.

Low impact / low likelihood issues are the lowest priority and present the smallest threat.

See Appendix for more information.

Rating	Category	Test Type	Instance Count
Critical	Cross-Site Scripting: DOM	Static	7
Critical	Cross-Site Scripting: Reflected	Static	17
Critical	Password Management: Hardcoded Password	Static	9
Critical	Password Management: Insecure Submission	Static	2
Critical	SQL Injection	Static	71
High	Password Management: Empty Password	Static	1
High	Password Management: Hardcoded Password	Static	6
High	Privacy Violation: Autocomplete	Static	2
High	Privacy Violation	Static	48
High	SQL Injection	Static	352
Low	Cross-Site Request Forgery	Static	29

Figure 28: Category of issues reported after preliminary scan

The file instances that have issues and should be modified are mentioned below:

BadgeBrowser.php	earner.php	GetBadges.php	BadgeMaker.php
badgeManager.php	Badge_Issued.php	Fg_membersite.php	Change-pwd.php
login.php	Badges.php	User.php	myBadge.php
register.php	GetBadges.php	Pwdwidget.js	ProfilePage.php
Group.php	GetCounties.php	Jquery-1.10.1.min.js	Upload.php
youthbadge.php	GetUser.php	Class.phpmailer.php	Reset-pwd-req.php

The functionalities of these files are to log in with 4-H account, user registration, browse a badge, badge management, earn a badge, reset password, display my badges, create a badge, change password.

6.3.2 Suggestions or Recommendations

Cross-Site Scripting Vulnerability

For XSS attacks, Web application must validate their input to prevent vulnerabilities. The most secure approach to validation for XSS is to create a whitelist of safe characters that are allowed to appear in HTTP content and accept input composed exclusively of characters in the approved set. For example, a valid username might only include alpha-numeric characters or a phone number might only include digits 0-9. A more flexible, but less approach is known as blacklisting, which selectively rejects or escapes potentially dangerous characters before using input.

Another approach to preventing XSS attacks is encoding. Server-side encoding is a function in which scripting tags in all dynamic content can be replaced with codes in a chosen character set. The J2EE, Struts and webworks frameworks support encoding.

Password Management Vulnerability

For Password Management attacks, passwords should never be hardcoded and should generally be obfuscated and managed in an external source.

SQL Injection Flaws

SQL queries uses invalidated user input as vulnerable script. In an SQL injection attack, an attacker sends malicious input through the application interface like login page and these queries are executed in the backend database and attacker can gain control over a database or perform DOS attacks like Database shutdown. The following example illustrates JSP login form where the attacker can enter malicious commands like “XP_cmdshell” or “*; OR 1=1 —“etc.

```
String Uname = request.getParameter (“User”);
String Pword =request.getParameter (Password”);
String s = “SELECT * FROM Table WHERE Username = ‘ “ + UName + “ “ AND Password = ‘ “ + Pword “ ‘ “;
Statement stmt = connection.createStatement ();
ResultSet rs = stmt.executeQuery (s);
If (rs.next ())
{
UID = rs.getInt (1)
User = rs.getString (2)
}
PrintWriter writer= response.getWriter ();
writer.println (“User Name: “+ User);
}
```

In the above example, the SQL query is created using the username and password transferred to the server without validating the input which is vulnerable to a SQL injection attack. To mitigate the risk of SQL injection is to use only stored procedures or parameterized database calls. Using prepared statement, all the queries will be treated as a string but not commands to be executed by the database. The following code can be used as remediation for the above vulnerable code.

```
String s = “SELECT * FROM Table WHERE Username = ‘ “ + UName + “ “ AND Password = ‘ “ + Pword “ ‘ “;
PreparedStatement stmt = connnection.prepareStatement(s);
stmt.setString (1, UName);
stmt.setString (2, Pword);
ResultSet results = stmt.execute ();
```

Privacy Violation

For Privacy violation: Autocomplete, Explicitly disable autocompletion on forms or sensitive inputs. By disabling autocompletion, information previously entered will not be presented back to the user as they type. It will also disable the "remember my password" functionality of most major browsers.

Cross-Site Request Forgery vulnerability

For Cross-Site Request Forgery, applications that use session cookies must include some piece of information in every form post that the back-end code can use to validate the provenance of the request. One way to do that is to include a random request identifier. The back end logic can validate the request identifier before processing the rest of the form data. The request identifier can be unique to each server request or can be shared across every request for a particular session. As the session identifiers, the harder it is for an attacker to guess the request identifier, the harder it is to conduct a success CSRF attack.

6.3 Work Plan

"For Youth For Life" system should implement following techniques in order to prevent security vulnerabilities:

1. *Data Validation*: Different types of input that can be modified by a malicious user such as HTTP headers, input fields, hidden fields, drop down lists and other web components should be properly validated. This can be performed with respect to a whitelist of allowed characters. Proper length check should be performed on all input exists.

   It could lead to Injection flaws such as SQL, OS and LDAP occur when untrusted data is sent to interpreter as part of a command or query. Cross-Site-Scripting XSS occurs whenever an application takes untrusted data and sends it to a web browser with our proper validation.

2. *Authentication and Authorization*: FYFL system should clearly define the user types and the rights of said users. All the internal and external connections (user and entity) should go through an appropriate and adequate form of authentication. This control is should not be bypassed. Whenever authentication credentials or any other sensitive information is passed, only the information via the HTTP "POST" method should be accepted, no information is accepted via the HTTP "GET" method.

   A Cross-Site Request Forgery (CSRF) attack forces a logged on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows attackers to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

3. *Session Management:* The session ID should be examined and verified if it is complex enough to fulfill requirements regarding strength. Examine how sessions are stored and how the application tracks sessions. Also, examine the actions that the application takes if an invalid session ID occurs is determined. Session HTTP inactivity timeout should be determined.

4. *Error Handling:* All method/function calls that return a value should have proper error handling and return value checking. Exceptions and error conditions should be properly handled. No system errors should be returned to user and application should fail in a secure manner. Resources should be released if an error occurs.

7. Experiment Design

7.1 Participants

Participants were students from the FYFL system development team. They had a background in using computer and knowledge in code security. At least two to three persons were experts in the area of code security. This helped performing a heuristic review and cognitive walkthrough. This ensured that study have been viewed from different angles and to ensure the quality of our results. The study was performed at Auburn University.

Participants drive awareness of SSA solution by securing support from key stakeholder. They communicate the business value of software security, set aggressive goals for applications and developer coverage, and invest in software security education and training.

7.2 Process

Driving the vulnerability-prevention processes like input validation deeper into the development of the FYFL system and systematically prioritize vulnerabilities to focus remediation plans. It requires code scans at strategic checkpoints in the development process such as during nightly builds or before releasing applications to production. Also, requires rapidly integrating software security resources with development teams, including software security performance, adopting of SSA practices, and track their compliance. This is a white box testing, as we run through the code to understand the functionalities of the system to identify the security flaws.

7.3 Verification and Validation

Quality assurance especially for software is a complicated one. Our process of Quality assurance includes:

1) Conformance with standards

2) Configuration Management - a controlled and documented change

3) Verification and Validation - a defense against error prone software development and maintenance process

4) Test and Evaluation - software code exercise and assessment

The discipline of verifying that software products of each Software Development Life Cycle phase comply with previous life-cycle phase requirements and establish the proper basis for initiating the next life cycle phase, and of validating that the completed end product complies with the established software and system requirements.

Verification: “Are we building the product right”

Validation: “Are we building the right product”

To make sure that we are building the product right, we performed code traceability analysis and code evaluation as a group while implementing the input validation techniques to detect the defects. During this phase, the source code, the executable code, the user documentation and standards are taken as inputs. The detailed issue report and summary report are the outputs. We implemented a verification and validation technique named “Inspection”, which is a formal code evaluation in detailed by a person other than author to detect the security vulnerabilities.

The following describes our process for code traceability and evaluation of the file "Badges.php". The file gets badges from database and filter by search parameters (tag, state, county) to return as an array. The tag value is passed as an input to method getBadgesFromTags(), which returns the relevant badges. The post variable's state, county and tag do not have any data validation implemented which might cause serious security issues. We have added a regular expression based whitelist input validation for variables tag, state and county which do not affect the functionality of the file.

```
<?php
    require_once(__DIR__ . "/dbc.php");
    require_once(__DIR__ . "/tags.php");
    require_once(__DIR__ . "/include/membersite_config.php");
/*Get badges from database and filter by search parameters to return as
 *an array.
 */
    function getBadges($offset,$limit,$tag,$state,$county)
    {
        global $dbc;
        $matches = preg_match('^[a-zA-Z0-9 ]*[.'-]?[ a-zA-Z0-9]*[.'-]?[ a-zA-Z0-9]*', $tag);
        if($matches!= 0){
    //get all the badges with the given tag
        $badges=getBadgesFromTags($tag);
    //get all badges
        $q="SELECT image_id,badge_name,badge_desc,status FROM images3 ";
        $whereAnd="WHERE";
    //filter by tags
        if(count($badges)>0)
        {
            $q.=$whereAnd.' image_id IN ('.implode(',',$badges).') ';
            $whereAnd="AND";
        }
    //filter by state
        if(!empty($state))
        {
        if (ctype_alpha($state)) {

$q.=$whereAnd.' state="'.$state.'" ';
```

Figure 29: Code Tracing and Code Evaluation

Also, verified the software product after implementing the input validation techniques by scanning the entire code base through "HP Fortify on Demand" to see if it was properly implemented and has mitigated the security vulnerabilities. Considering an executable software product code as benchmark and improving the security of the code without changing any

functionality of the system during the development phase, validates that end product complies with established software and system requirements in previous phases.

The benefits of performing verification and validation are reduction in number of vulnerabilities identified after each scan. Also, increases the productivity of the system through early detection and mitigation of security vulnerabilities.

7.4 Data Collection & Analysis

After performing the preliminary scan of the code, results obtained are analyzed. In total, there are 544 issues identified of which severity of 106 issues is critical, 409 issues is high, and 20 issues low.

SQL injections have a high impact. In order to mitigate the SQL injection issues, we have implemented data validation techniques depending on the variable and input value in that particular file. In the study, fixing SQL injections involved:

- Changing the SQL queries in the files “Badges_Issued.php”, “Badge_Issued_Comp.php”, “Badge_Issued_Plat.php”, “Badge_Issued_mech.php”, “Badge_Issued_Mov.php” etc. to prepared statements and parameterized queries as variables iname, fname, bname can be tampered. These SQL statements are now sent to and parsed by the database server separately from any parameters. By specifying the parameters either a “?” or a named parameter like “:name”, you tell the database engine where you want to filter on. Any parameters you send when using a prepared statement will be just be treated as strings. Then when you call execute, the prepared statement is combined with the parameter values you specify.

```
//          mysql_query("SET CHARACTER SET utf8",$dbc);
        }
          $id=$row['id_user'];

            $stmt= $dbc->prepare ('UPDATE Hands_Response Set action='c' where id_user=?');
$stmt->bind_param('s', $iname);
$stmt->execute();
            $r = $stmt->get_result();

           // $q = "UPDATE Hands_Response
            //Set action='c'
            //where id_user='$iname'";


            // Execute the query:
            //$r = mysqli_query ($dbc, $q);


        // Check the results:
        if (mysqli_num_rows($r) == 1) {

            // Retrieve the image information:
            $row = mysqli_fetch_array ($r, MYSQLI_ASSOC);

          // Clean up:
```

Figure 30: SQL Injection fix 1

- Performing input validation for the $_post variables state, county, tag in files “Badges.php”, “GetBadges.php”, “tags.php”, “GetCounties.php” etc. According to the definition, value of the “state” variable always contains only two alphabets. So, we perform a check of value of the state using a PHP Boolean method ctype_alpha( ). According to the definition, value of the “county” and “tag” variables contain only alphanumeric. So, we perform a check of values of the variables using a regular expression by white listing the characters and the regular expression looks like “(^[a-zA-Z0-9]*[.'-]?[a-zA-Z0-9]*[.'-]?[a-zA-Z0-9]*)”. The PHP method used to perform the match of the value and regular expression is preg_match( ).

```php
<?php
    require_once(__DIR__ . "/dbc.php");
    require_once(__DIR__ . "/tags.php");
    require_once(__DIR__ . "/include/membersite_config.php");
/*Get badges from database and filter by search parameters to return as
 *an array.
 */
    function getBadges($offset,$limit,$tag,$state,$county)
    {
        global $dbc;
        $matches = preg_match('^[a-zA-Z0-9 ]*[.'-]?[ a-zA-Z0-9]*[.'-]?[ a-zA-Z0-9]*', $tag);
        if($matches!= 0){
    //get all the badges with the given tag
        $badges=getBadgesFromTags($tag);
    //get all badges
        $q="SELECT image_id,badge_name,badge_desc,status FROM images3 ";
        $whereAnd="WHERE";
    //filter by tags
        if(count($badges)>0)
        {
            $q.=$whereAnd.' image_id IN ('.implode(',',$badges).') ';
            $whereAnd="AND";
        }
    //filter by state
        if(!empty($state))
        {
        if (ctype_alpha($state)) {

$q.=$whereAnd.' state="'.$state.'" ';

       $whereAnd="AND";
```

Figure 31: SQL Injection fix 2

- Sanitizing the $_post variable “username” read from login( ) method in fg_membersite.php before it is being passed to methods in files “GetUser.php”, “User.php”. White-listing the characters using a regular expression and performing a check before the username is passed to the SQL statement.
- Performing an input validation for the $_get variable “groupid” and the $_post variable “groupID” in the methods usersfromgroup(), getgroupinfo(), pairusertogroup() in Group.php are read from groupListing.php and test.php. According to the definition, value of groupid consists of only numbers. Therefore, we perform validation of groupid using Boolean PHP functions is_numeric() and ctype_digit().

```
    else{
        $username = $userInfo['username'];
        $firstName = $userInfo['name'];
        $lastName = $userInfo['lname'];
        $email = $userInfo['email'];
        $role = $userInfo['Adult'];
        $user_level = $userInfo['user_level'];
        $userId = $userInfo["id_user"];
        $State = $userInfo['State'];
        $County = $userInfo['County'];
        $groups = groupsFromUser($userId);
    }


    if(isset($_GET["groupId"]) && is_numeric($_GET["groupId"])){
        $groupId = $_GET["groupId"];
    }
    else {
        $groupId = 11;
    }
?>
```

Figure 32: SQL Injection fix 3

```
<?php
    require_once(__DIR__ . "/Group.php");
    require_once(__DIR__ . "/User.php");
//get the current user's info
    $user=getUser();
//make sure the user is logged in (not false) before pairing with group
    if(user!=false&&isset($_POST['submit']) && ctype_digit($_POST['groupID']))
    //pair the current user's ID with the given group's ID
        pairUserToGroup($user['id_user'],$_POST['groupID']);
?>
```

Figure 33: SQL Injection fix 4

In the study, fixing Privacy Violation: Autocomplete involved:

- With Auto completion enabled, some browsers retain the sensitive information (user input) in their history, which allows someone using the computer after the initial user to see

information previously submitted. By default, forms set autocompletion “on”. So fixing privacy violation involves setting forms in “fg_membersite.php” to autocompletion = “off”.

Figure 34: Privacy Violation- Auto completion fix

In the study, after analyzing the Password Management issues, we found that they were not really the passwords and so the issues are marked as false positives. If they are real passwords, then the study recommends using PHP methods mcrypt_encrypt and mcrypt_decrypt for encryption and decryption of the plaintext passwords.

7.5 Expected Outcomes

We conducted an exploratory analysis of the effectiveness in conducting the security code review by implementing the data input validation techniques. We predicted the effectiveness of the techniques implemented by observing, the reduction in number of issues in comparison with the initial findings.

Our initial findings provided us with the opportunity to find how many vulnerabilities were found and what specific vulnerabilities were found most and least commonly. After fixing the issues of SQL injection and Privacy violation, we observe that the issues are mitigated completely. Therefore, it proves the effectiveness of the techniques implemented.

7.6 Experiment Results

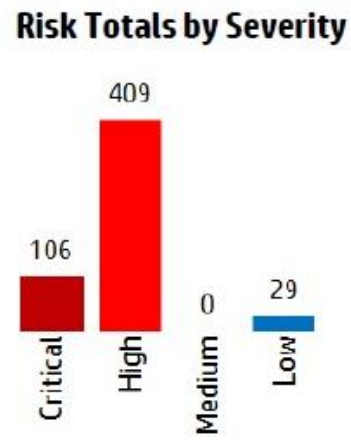
After the initial scan of code, there were 71 Critical SQL injections and two Privacy Violation: Auto completion vulnerabilities identified in 24 source code files. We implemented the whitelist input validation technique to fix SQL injections and Privacy violation identified in six files taken as a sample and rescanned the code. We observed that 15 SQL injections and two privacy violation were mitigated. Below figure represents the re-scan results:

Rating	Category	Test Type	Instance Count
Critical	Cross-Site Scripting: DOM	Static	7
Critical	Cross-Site Scripting: Reflected	Static	17
Critical	Password Management: Hardcoded Password	Static	9
Critical	Password Management: Insecure Submission	Static	2
Critical	SQL Injection	Static	56
High	Password Management: Empty Password	Static	1
High	Password Management: Hardcoded Password	Static	6
High	Privacy Violation	Static	48
High	SQL Injection	Static	352
Low	Cross-Site Request Forgery	Static	29

Figure 35: Category of issues after 2nd scan

From the results, it is observed that whitelist input validation implemented mitigates the SQL injections better than blacklist validation method "sanitize" being implemented earlier. Blacklisting is the process of validating a desired input against a list of negatives inputs. It is basically blocking all the unwanted characters. Whitelisting validates a desired input against a list of possible correct inputs. An attacker may use any means possible to gain to access to the web based application and we cannot eliminate all possible bad conditions. Whitelist is the best way to validate the input as we know exactly what is desired as an input. As the best way to implement the whitelist validation is using regular expressions, we have used regular expressions to validate username, state name, county etc.

As the 2nd scan proved whitelist validation is a better way to validate the input, we implemented this technique in 13 files identified having SQL injections. We performed a 3rd scan of the entire source code and observed that 25% of the total issues i.e. 71 critical severity SQL injections, two high severity Privacy violation were mitigated and 11 Password Management were found to be false positives (i.e. not really passwords). Bar graphs below are used to depict the change in count of vulnerabilities (106 to 35 critical and 409 to 407 high) from the initial scan to 3rd scan by HP Fortify on Demand.



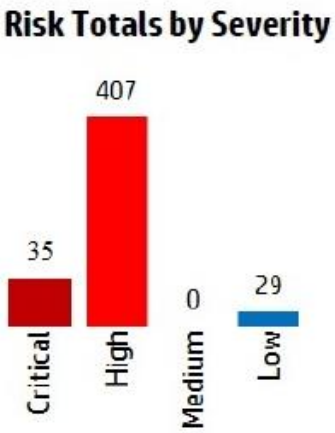


Figure 36: Results of Initial scan *Figure 37: Results of third scan*

Below mentioned is a table of performance metrics:

Performance Metrics	
Percentage of Vulnerabilities mitigated	25% of vulnerabilities identified (103 critical issues of 544 total issues)
Average time to identify a vulnerability	Manual testing took 1 to 2 weeks and Automated testing took 1 to 2 hours
Average time to fix a vulnerability	Takes from 1 to 2 days
Percentage of number of source files fixed	60% of the numbers of files identified having vulnerabilities (i.e. 14 out of 24)

8. Summary and Conclusion

As the security threat moves from computer-network intrusions to attacks on software applications running in multiple environments, the demand for software security assurance solutions is on the rise. SSA Solution (HP Fortify) helps the FYFL system minimize the risk of a successful cyber-attack. It also offers substantial efficiency and production benefits that help speed software development cycles. Most current techniques for detecting web security vulnerabilities are automated tools for static analysis. The study discussed various commercial automated tools available and also comparison of different tools.

The study illustrated the value of identifying the security risks existing in the FYFL using an SSA solution (HP Fortify). The study also analyzed the severity of risks, implemented and accelerated correction of the security vulnerabilities. Some of the fixes proposed like whitelist input data validation etc. can be exponentially higher. More benefits can be generated by extending the solutions, by embedding security controls and best practices throughout the development lifecycle. Therefore, every developer should be required to do a threat analysis before writing the first line of code.

The significance of the study is to identify the key techniques to be implemented while programming to prevent the security vulnerabilities, to identify the vulnerabilities currently existing in the system, and to improve the security of the FYFL system to the gain customer trust by implementing the whitelist data input validation techniques to mitigate the risks.

Bibliography

[1] Adrian Fernandez, Emilio Insfran, and Silvia Abrahão. 2011. Usability evaluation methods for the web: A systematic mapping study. *Inf. Softw. Technol.* 53, 8 (August 2011).

[2] Marc Hassenzahl. 2008. The interplay of beauty, goodness, and usability in interactive products.*Hum.-Comput. Interact.* 19, 4 (December 2008).

[3] David Scott and Richard Sharp. 2002. Abstracting application-level web security. In *Proceedings of the 11th international conference on World Wide Web* (WWW '02). ACM, New York, NY, USA.

[4] Elfriede Dustin, Jeff Rashka, and Douglas McDiarmid. 2002. *Quality Web Systems: Performance, Security, and Usability*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[5] Kiran Maraju. Security Code Review – Identifying Web Vulnerabilities

[6] HP CloudSystem Enterprise. Integrating security with HP Fortify

[7] Anne Edmundson, Brian Holtkamp, Emanuel Rivera, Matthew Finifter, Adrian Mettler, and David Wagner. 2013. An empirical study on the effectiveness of security code review. In *Proceedings of the 5th international conference on Engineering Secure Software and Systems* (ESSoS'13), Jan Jürjens, Benjamin Livshits, and Riccardo Scandariato (Eds.). Springer-Verlag, Berlin, Heidelberg.

[8] Yong Chen and Wu He. Security risks and protection in an online learning – A survey

[9] Nikhilesh Barik, Argha Barik, Dr.Sunil Karfoma. 2012. On Security Management in E-Learning System

[10] A white paper. Does Application Security Pay? Measuring the business impact of software security assurance solutions.

[11] Andres Desa.2005. Document Security in Web applications (OWASP).

[12] Andrew van der Stock, Dinis Cruz, Jenelle Chapman, David Lowery, Eoin Keary, Marco M.Morana, David Rook, Jeff Williams, Paulo Prego. OWASP (Open Web Application Security Project) Code Review Guide and OWASP Secure Coding Practices.

[13] OWASP Foundation. (2002-2008). *OWASP testing guide V3*. Retrieved January 16th, 2015, from https://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf

[14] Napolean – Alexandru Sireteanu. December 20, 2008. Improving the Usability of Web Applications.

[15] Kyle Sollenberger. (2012, August 7). *10 User Interface Design Fundamentals.* Retrieved from http://blog.teamtreehouse.com/10-user-interface-design-fundamentals

[16] *User Interface Design Basics.* Retrieved from http://www.usability.gov/what-and-why/user-interface-design.html

[17] *User Interface and Future Interaction Technologies.* Retrieved from https://www.ida.gov.sg/~/media/Files/Infocomm%20Landscape/Technology/TechnologyRoadmap/UserInterface.pdf

[18] Peter Vukovic. (2015, February 13). *7 user interface design trends you need to know about.* Retrieved from https://99designs.com/designer-blog/2012/06/20/7-user-interface-design-trends-you-need-to-know-about/

[19] IBM Software. (2014, January). *Manage application security risks to help protect your organization's critical data*.

[20] Gary McGraw, Cigital. (2008, December). Automated Code Review Tools for Security

[21] Natarajan Meghanathan. (2013, January). Source Code Analysis to Remove Security Vulnerabilities in Java Socket Programs: *A Case Study*