

# **Security Lab Manager: Virtual Security Training for Universities**

**Simon Owens**

**Computer Science**

**University of Evansville**

**December 2018**

## **ABSTRACT**

The Security Lab Manager is a web application that manages vulnerable virtualization machines that allows users to experience security issues on. Users can login, choose an exercise, and start hacking in their own virtual environment. Administrators can login to view completed exercises and send grades to users. New exercises and machines can easily be created or imported.

## LIST OF FIGURES AND TABLES

*Figure 1 – Host Architecture*

*Figure 2 – Login Page*

*Figure 3 – Student View*

*Figure 4 – Administrator View*

*Figure 5 – Development and Deployment Process*

*Figure 6 – Vulnerable JavaScript*

*Figure 7 – Example XSS*

*Table 1 – User*

*Table 2 – Administrator*

*Table 3 – Class*

*Table 4 – Exercise*

## INTRODUCTION

Learning modern security practices is difficult and time consuming. Much of this time can be spent setting up safe environments to practice and reproduce vulnerabilities in. Some security exercises may not work because of different operating system protections, configuration settings, permissions, and patch version. The Security Lab Manager eliminates these problems by having a variety of exercises that can be started in seconds. This application has a convenient interface that allows users to start, stop, and restart exercises, and submit answers when they are complete.

Users can practice exploiting a variety of web and desktop C++/Java applications. For each exercise, there are guides for how to code securely and prevent the vulnerabilities just exploited. Administrators can create multiple choice questions to introduce certain topics before doing hands on exercises. Administrators can create their own exercise or import a vulnerable machine.

Static analyzers [[Appendix A](#)], Vulnerability Scanning [[Appendix B](#)], CI/CD [[Appendix C](#)], and test driven development [[Appendix D](#)] were used for the creation of this application. This makes the application production ready – by ensuring for few vulnerabilities and a secure design.

Easily updatable – by using a Jenkins pipeline to ensure updates are frequent and don't break functionality. Easily deployable – a Windows and Linux build script that uses Docker containers [[Appendix E](#)] allows for easy installation to a desktop, on a server, or on a Cloud Computing platform.

## PROBLEM STATEMENT AND BACKGROUND

Universities desire to teach software security because of the industry demand for secure coding and Security Engineers. The best way to prepare students is with hands-on experience seeing, exploiting, and patching vulnerabilities. Setting up a practice area for students with multiple computers is expensive and requires management. The typical setup would depend on the class size and available funds: running vulnerable virtual machines would not be able to support a class size of hundreds of students or resources could be wasted if too much infrastructure was allocated. Services would have to be setup, systems updated, and users would have to be added/removed. Students will frequently crash their target computers which requires constant troubleshooting and resetting. If students are allowed full permission to the infrastructure to troubleshoot their own problems, they could do nefarious things or even break the infrastructure.

There are a variety of problems when students are responsible for setting up their own environment and exercises. These exercises require virtualization software to run on because of software compatibility issues and risk of harming the student's computer. Students could host their own virtual machine, but this takes time away from class, requires computing power, and does not give students unique answers to submit. Setting up a victim and attack virtual machine takes several hours to do and does not directly help students learn security. Just getting one exercise to work might require installation and configuration of: an operating system patch, a DLL, a library, an application, networking, firewall settings, registry settings, and anti-virus rules. This configuration usually requires 4GB of RAM, and a couple CPU cores on top of the student host OS. This which may be impossible for some students or result in an extremely slow experience for others. Vulnerable machines from the Internet also do not have unique answers,

so one student could do the exercise, email it to the rest of the class, and the instructor would have no way of knowing who did the exercise.

Even if all of these efforts were planned, supported, and managed there are not any solutions that translate student exercises to grades for professors. Professors could take the time to create many exercises and vulnerable virtual machines but there are already hundreds of great resources available on the Internet. This is where this project comes in – the Security Lab Manager. It takes these vulnerable exercises others have already made and manages them so students can attack, destroy, and reset. Professors view how long students spent on their exercises, and all of the commands they performed. If the class is not ready for hands on exercises, the instructor can easily create their own multiple choice exercises for students to complete. Hosting this application takes minimal resources and can scale easily to the class size. The GUI and exercises will work seamlessly for all class sizes. Professors have a nice interface to view completed student exercises and be notified if any students cheated.

## REQUIREMENTS AND SPECIFICATIONS

These requirements and specifications deliver the functionality that professors and students need in order to learn security at a rapid pace. The main goal of this application is to deliver a secure portal to professors and students to interact with virtual machines.

1. GUI interface for students to login, launch exercises, revert machines, and submit answers

This GUI will have two main components: a grading page for professors and a page for students to interact with their exercises.

2. GUI interface for teachers to login and view answers of students

This interface should display which students have submitted answers and if any of their answers match each other. Since each student should have a unique answer, this will catch cheating.

Professors should be able to assign grades within seconds.

3. Students should be able to start, stop, cancel, and revert their security exercises

As a student, they should always know what action is currently processing, and have the ability to cancel.

4. The application should allow a student to launch only one exercise at a time

This limits the resources the application consumes. Students should work on only one exercise at a time, so they should be restricted by the application.

5. The application should be multi-threaded

Users should never have to wait for server-side action to complete before issuing other actions.

This makes the application feel nice and smooth.

6. There must be at least 3 web security exercises

This allows users to immediately start practicing upon download. No additional configuration needs to be done in order to start learning. Web security is extremely relevant in today's industry.

7. There must be at least 3 desktop application security exercises

This allows users to immediately start practicing upon download. No additional configuration needs to be done in order to start learning. Desktop application security is still relevant but less common in security jobs.

8. The application must be developed securely with static analyzer and must undergo scanning from web application scanning tool

Students that learn more about security may be tempted to try attacking this application for fun or to even change their grade. OWASP ZAP [[Appendix B](#)] will help detect vulnerabilities during each build.

9. This application should be extremely easy to setup, updated, and have documentation

Administrators should only have to download and run one command to install the application. Administrators get reports on any issues, vulnerabilities, and code quality on the download page.

10. Each exercise must be uniquely identified for each student.

This helps translate security exercises into grades for students. This feature helps prove that the student did their own work.

11. There must be an proxy in front of the application for scalability

Some environments may have hundreds of students which could make the web application slow.

Using Nginx allows for static files to be delivered faster, and allows administrators to spin up more applications to meet the amount of users.



## DESIGN

### Overview

This will be a Django [[Appendix F](#)] project that interfaces with Docker to launch virtual machines. Figure 1 show the architecture for how users can login and reach exercises.

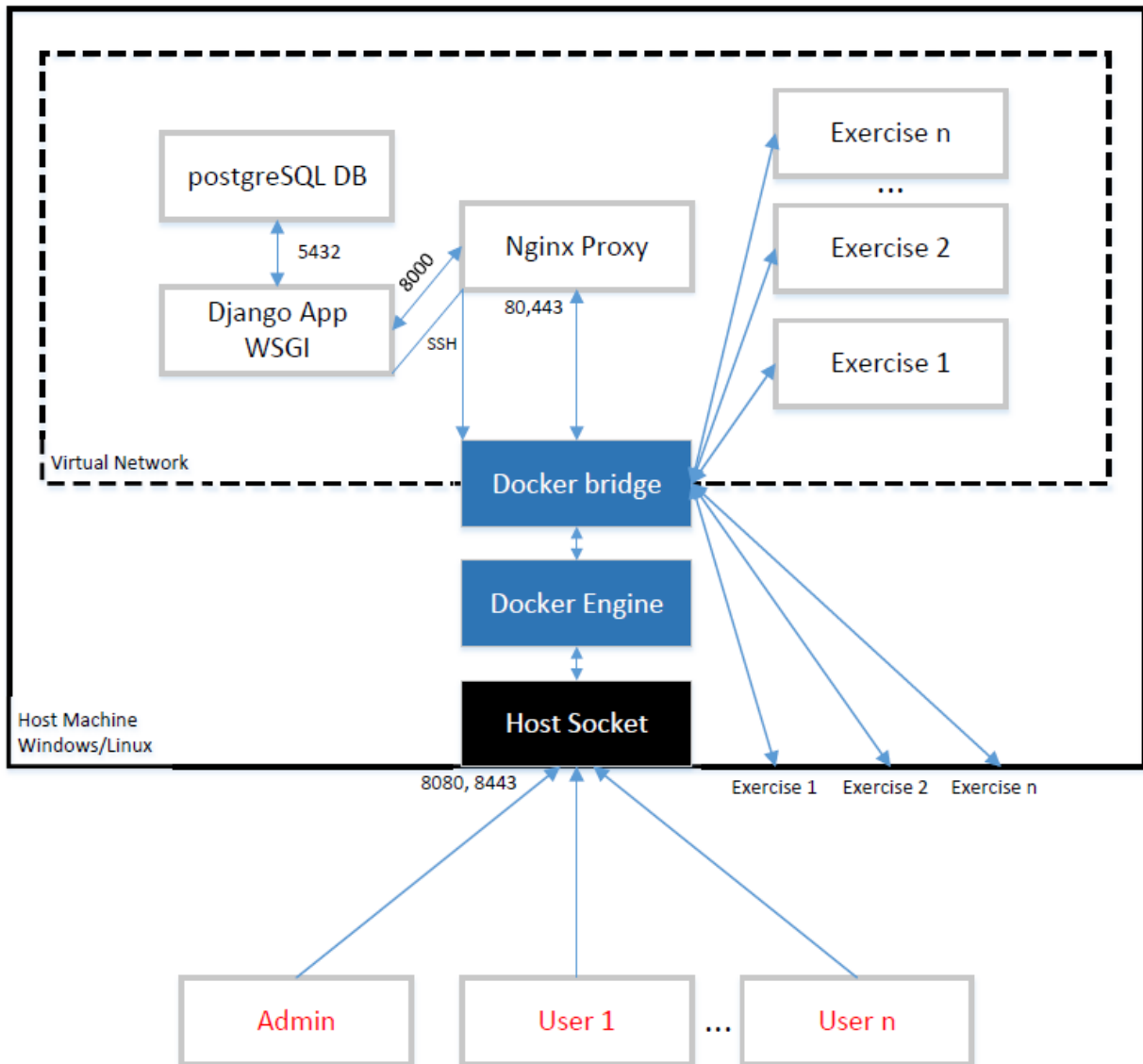


Figure 1 – Host Architecture

The Security Lab Manager is a collection of Docker services working together to virtualize this environment: a proxy, a web front-end, a back-end, and a database. Docker containers are used to eliminate installation compatibility issues, scalability, and because they are lighter weight than other virtualization software. An administrator can download the project and install the application with one click on either Windows or Centos7 running Docker - the installer only needs to enter the master password for the application. The administrator can then visit the IP of the host computer via HTTPS to login and start creating users.

Once students login, they will be able to view various exercises and start them. Starting an exercise will launch a light-weight Docker container. This container will have a unique hash in the root directory based on: the teacher's password, student's name, and exercise name. The goal is for students to find the vulnerability, exploit it, and then find the unique hash in the exercise. Once students complete the exercise, they can submit their unique hash to the application. If students crash the virtual machine, they can simply restart it with one click.

Administrators can then view student's progress and be alerted if any hashes submitted are the same. If administrators wish to add any new exercises, they can create a multiple choice question or create their own vulnerable virtual image. They can then import that vulnerable image into the application by entering: the exercise's name, where it should be grouped, and the Docker image name.

What are some potential issues for users?

- Students will be sending malicious traffic across the network at this Security Lab Manager. This could potentially violate any University policies.

- This application can launch Docker containers with full permissions. If the main application was compromised the attacker could use resources of the host machine and pivot onto other targets.
- The Security Lab Manager must be centrally hosted and have computing power to support the class size

### *Graphical Interface*

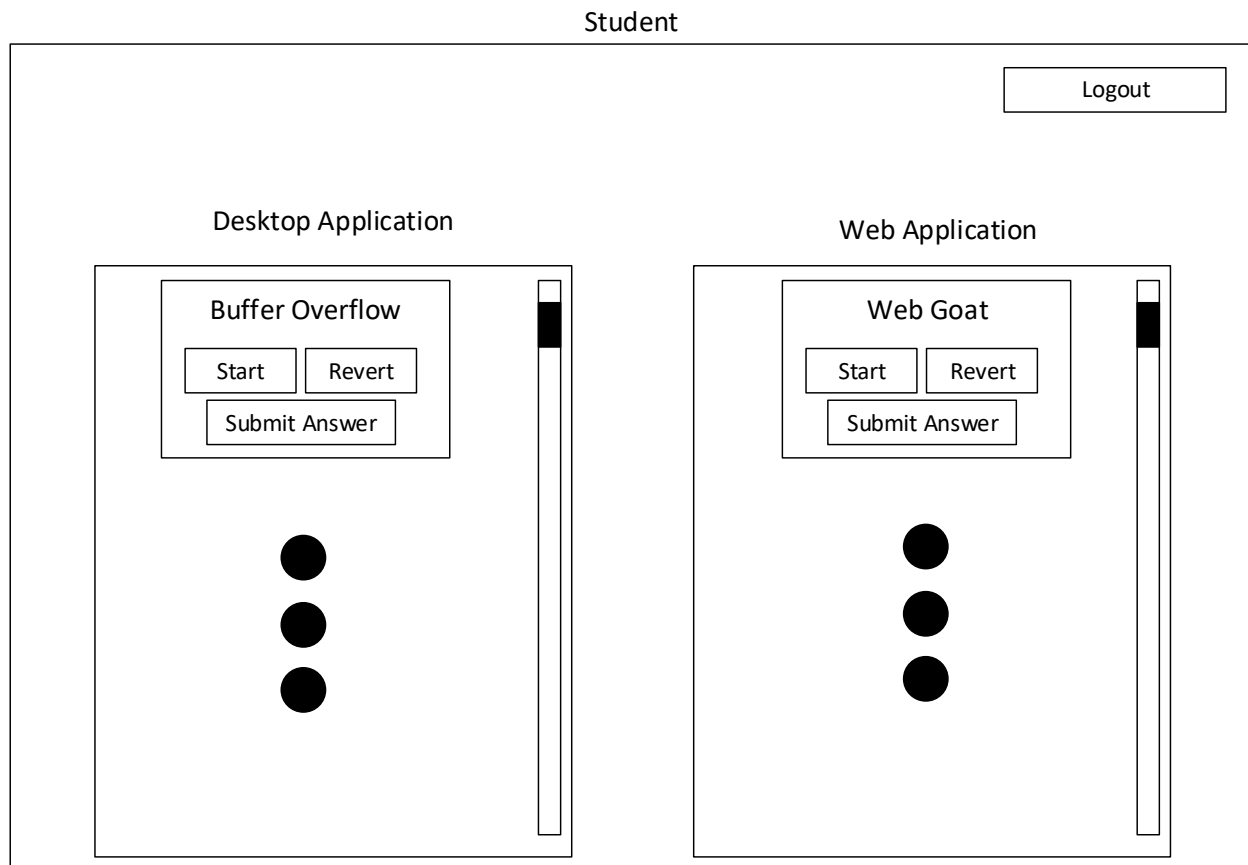
The login page show in Figure 2, is the same for users and administrators.



The figure shows a login page for the Security Lab Manager. It features a large, bold title "Security Lab Manager" centered on the page. Below the title are two input fields: the top one is labeled "Username" and the bottom one is labeled "Password". Both fields are rectangular with thin borders. The entire login form is centered within a larger rectangular frame.

*Figure 2 – Login Page*

Users will be directed to the page shown in Figure 3 where they can launch exercises and submit their answers. Users can see all of the different sections, with all of the exercises associated with them.



*Figure 3 – Student View*

Administrators have an entirely different view shown in Figure 4 – they can manage the performance of the application and see which students completed their exercises.

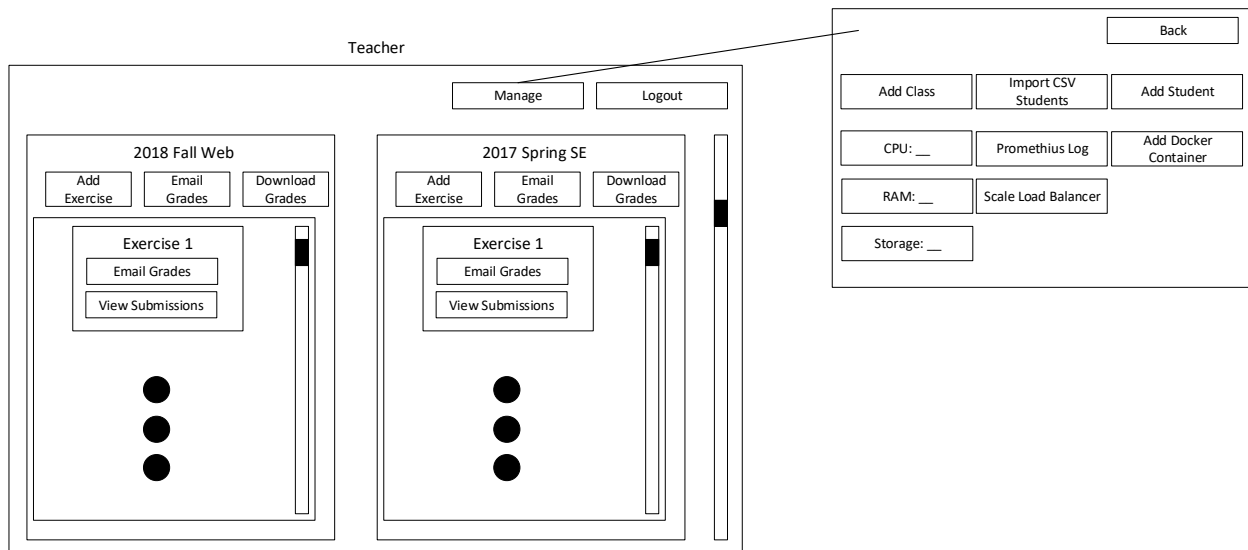


Figure 4 – Administrator View

They can easily scale the application, add users, and send out grades to students.

### Database

Information for users, administers, classes, and exercises will need to be stored. A PostgreSQL database will be used because of my familiarity with the database already – it works with Django, is fairly fast, and has a low learning curve. Information for users will be stored in Table 1.

DB Attribute	Description
Name	Identifies in human readable way
Password hash	For login and unique hash in exercise
Email	Unique and allows for communication
Classes<Array>	The classes the user has access to

Table 1 – User

Administrators will have a different table since they have access to all exercises shown in Table 2.

DB Attribute	Description
Name	Identifies in human readable way
Password Hash	For login and unique hash in exercise
Email	Unique and allows for communication

Table 2 – Administrator

Each class will be comprised of various exercises shown in Table 3.

DB Attribute	Description
Class Name	Identifies in human readable way
Exercises<Array>	This list of exercises belonging to a class

Table 3 – Class

Each exercise should have a unique hash for every user shown in Table 4.

DB Attribute	Description
Name	Identifies in human readable way
Configuration	Text for multiple choice or container
Answer Hash	Unique for every user and exercise

Table 4 – Exercise

*Vulnerable Exercises*

Three custom web and desktop exercises will be created. There will also be instructions on how to create and import new exercises into the application. One of the web exercises will be a cross-site scripting vulnerability. This exercise will be built on top of the Ubuntu Docker image. The example will be a JavaScript web page which contains the code in Figure 6.

```
<% String eid = request.getParameter("eid"); %>  
...  
Employee ID: <%= eid %>
```

Figure 6 – Vulnerable JavaScript

Users could then send the injected into the employee ID parameter like in Figure 7 to exploit the cross site scripting vulnerability.

```
<body onload=alert('test1')>
```

Figure 7 – Example XSS

There will be an administrator sending a POST to this page with their employee ID with their password as a cookie. Users can use the XSS to find the admins cookie and then login to the Ubuntu machine to retrieve the hash.

## DEVELOPMENT PLAN

### *Development Prerequisites*

#### *Agile – Sprint 1*

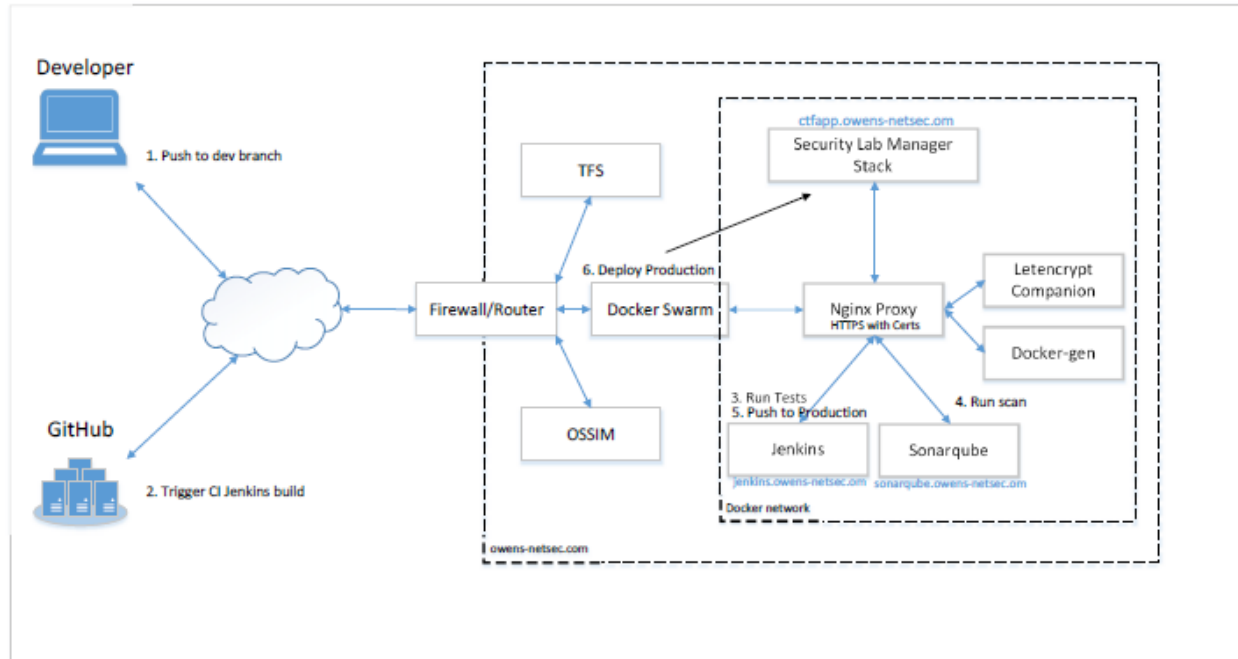
The goal of this senior project is to accomplish all of the requirements already laid out. The requirements may be removed or added once the velocity of development is understood. There will be a weekly demo on the new functionality added. Feedback will then drive the work for the next week until the end of development. Requirements will be broken down into tasks – which will be less than 4 hours and have a value weight associated with them. These tasks will be managed through Waffle IO on the GitHub repository page. All of the hours will be tracked in the engineering logbook located at [csserver.evansville.edu](http://csserver.evansville.edu).

Understanding the technologies and writing this proposal was the first step to accomplishing the requirements. The next step, to be completed in sprint 1, is to create the GUI prototype in HTML/CSS. This will ensure the product owner gets the look and feel they envisioned. The next step, completed in Sprint 2 or 3, would be allowing users and administrators to login, and click buttons.

### *Continuous Integration and Continuous Delivery*

This project is developed using CI/CD via Jenkins. [Jenkins](#) is a popular open source CI/CD tool. This allows the application to easily manage dependencies, vulnerabilities, and enables easy contribution. Figure 5 is a diagram for how the application gets developed and deployed.





*Figure 5 – Development and Deployment Process*

- Using Dock as the visualization image allows users to easily add new security exercises. I do not need to spend the time making new exercises since other professionals already make things like WebGoat, Bricks, and Damn Vulnerable Web Application [found on OWASP site](#).
- Using an Nginx proxy and Docker containers allows the administrator to scale the application's performance easily. This application could support anywhere from 5 to hundreds of users via load balancing and redundancy.
- The continuous integration Jenkins build will detect if a base container breaks functionality upon any update. A failed build on the development branch will not push to production so stable releases can always be used. Before any code can be added to production, all tests must pass, and there must not be any Sonarqube [\[Appendix A\]](#)

vulnerabilities, code smells, or bugs. Snyk and Dependabot [[Appendix B](#)] do scans against the project for common vulnerabilities and my dependencies.

- All requests to web application front-end come through Nginx via HTTPS so attackers cannot snoop on traffic or execute remote vulnerabilities easily since Nginx has a great security program.
- A vulnerability web scan is done against the system every build to ensure none of the OWASP top 10 exist in the web application.

## CONCLUSION

This application was developed in a way to maximize satisfaction from the product owner by presenting constant demos and gathering feedback. Modern development practices like test driven development, static analysis, vulnerability scanning, and CI/CD enhanced the applications security and stability. The Security Lab Manager is a great tool for learning security in a classroom setting safely.

## DEVELOPER CREDENTIALS

Before development of this application can start, a strong knowledge of CI/CD, Agile principles, Web Security, and the Django framework are required. These skills allow the secure delivery of what best fits the needs of the product sponsor's needs. In order to understand these practices, the following books have been read.

- Scrum – by Jeff Sutherland
- The DevOps Handbook – by Gene Kim
- Test-Driven Development with Python – by Harry J. W. Percival
- The Web Application Hacker's Handbook – by Dafydd Stuttard and Marcus Pinto

Concepts from the class Penetration Testing with Kali Linux, by Offensive Security, was used while testing this application for vulnerabilities. Finding vulnerabilities throughout development helped create vulnerable exercises because they were easily made mistakes by developers.

Simon Owens is a graduating from the University of Evansville in May 2019 at the age of twenty two years old. He has grown up in Evansville his entire life – but is working for Raytheon in Indianapolis as a Cybersecurity Engineer. Simon specializes in offensive security testing for Raytheon and working with developers on how to develop securely and integrate testing into daily workflow. His open source projects can be found at: <https://github.com/so87>.

## Appendix A – Static Analysis

Static Analysis is the method of analyze the syntax of a programming language for improper style and flaws without being ran. The analyzer will attempt to analyze logical paths and look for logic that could be exploited by certain input. Sonarqube was chosen as the static analyzer because of its popularity, free usage, IDE plugin, and support of Python, Javascript, HTML, and CSS. Sonarqube will display errors in the IDE while you code real time, and will give you project metrics like: total vulnerabilities, total code smells, test coverage, and total lines of code. Static Analyzers are used during development to decrease technical debt – because vulnerabilities are much cheaper to fix the faster they are found. Sonarqube will not catch all vulnerabilities, which is why vulnerability scanners are also used. To learn more about Sonarqube, please visit their [website](#).

## Appendix B – Vulnerability Scanning

OWASP ZAP – Vulnerability scanners send various inputs to a target and analyze the corresponding output for known vulnerabilities. There are vulnerability scanners for Operating systems, Docker containers, C++ applications, Web Applications, and so on. Since this project is a web application and uses various Docker containers, OWASP ZAP and Anchor Engine will be used. OWASP ZAP is a well-known web application vulnerability scanner that looks for weaknesses like cross site scripting, SQL injection, authentication bypass, and other common weaknesses. Each time a build is performed on this project, OWASP ZAP will begin a scan, and save the result to later analyze. This allows the developer to see what an attacker would see when scanning the application for vulnerabilities. Anchor Engine analyzes the contents of a Docker container for misconfigurations and vulnerable libraries/tools. An example is that some versions of SQL contain race condition vulnerabilities which can be exploited. If a Docker

container was running the old version of SQL, Anchor Engine would report this information after a scan. To learn more about [OWASP ZAP](#) and [Anchor Engine](#), please visit their websites.

## Appendix C – Continuous Integration and Continuous Delivery

Continuous Integration is the process of merging all approved code into a source controlled repository. Code is approved for integration if automated tests pass. This way all developers can improve the production code base without breaking functionality or injecting bugs. Continuous Delivery is the process of automatically making the integrated software changes to the production environment. This allows a developer to see their changes the same day in production – rather than upgrading application vulnerability every quarter or year. Jenkins is the Continuous Integration and Continuous Delivery (CI/CD) tool used for this project. Every time the development branch is updated, Static Analysis, Vulnerability scans, and tests are run. If all of those tests pass, and a certain level of quality is met, those changes are merged to the production branch, and then deployed on my local server. To learn more about CI/CD, or Jenkins, visit Jenkin's [website](#).

## Appendix D – Test Driven Development

Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: requirements are turned into very specific test cases, then the software is improved to pass the new tests, only. This is opposed to software development that allows software to be added that is not proven to meet requirements. Tests are written first, the test should fail, code is written to attempt to pass the test, once passed the code is reviewed, and this process is repeated for every requirement. This helps developers focus on meeting requirements and creating better tests that alert the developer when introducing a change breaks functionality.

Selenium and Mocha are used as the primary testing tools. Selenium allows for easy functional testing – a web browser is started, navigates to a certain page, and then looks for a specific result. Mocha tests are used for basic unit testing and to check for basic security configurations like redirects and key strength. To learn more about [Selenium](#) and [Mocha](#), please visit their websites.

## Appendix E – Docker Virtualization

Docker is a virtualization technology that puts applications and operating systems into what they call a container. This container is supposed to be a slimmed down version of a virtual machine – only the libraries and tools required to run an application are included. This generally results in a smaller and more efficient virtual machine. Docker has the ability to easily create multiple containers to scale to developer's needs. Docker has images on their [site](#) which are preconfigured for different applications. For more information about Docker, I recommend watching [this video](#).

## Appendix F – Django Framework

Django is a framework for creating websites. It is based on python and open source. It already has several built-in features to make authentication, scaling, and database interaction easy and secure. This is how websites - even simple ones designed by a single person - can still include advanced functionality like authentication support, management and administrator panels, comment boxes, file upload support, contact forms, and more. Python, JavaScript, HTML, and CSS are used in conjunction to create web pages for this project. To learn more about Django please visit their [website](#).

## REFERENCES