Spring 2019

Scalable Containerized Security Training Environment

Robert Sauer
Western Kentucky University, robert.sauer441@topper.wku.edu

SCALABLE CONTAINERIZED SECURITY TRAINING ENVIRONMENT

A Capstone Project Presented in Partial Fulfillment

of the Requirements for the Degree Bachelor of Science in Computer Science

with Honors College Graduate Distinction at

Western Kentucky University

By

Robert J. Sauer

May 2019

*****

CE/T Committee:

Professor Guangming Xing, Chair

Professor Zhonghang Xia

Professor Christopher Keller

ABSTRACT

The purpose of this project is to develop a portable application which is hosted on a server that provides an environment to safely conduct security training procedures and protocols. The project will be scalable to handle from a few to a multitude of users concurrently using a single server. For many users to perform security training simultaneously, each user must be directed to a sandbox environment, a container, where one user's actions do not affect the website or database of other users. Furthermore, such an application should be readily deployable into any environment to provide the widest range of compatibility. The anticipated outcome will be to provide a useful educational tool to enhance cybersecurity education with hands-on, interactive learning for schools and organizations.

Keywords: Security, SQL Injection, Docker, Containers, Apache

I dedicate this thesis to my parents, Chris and Theresa Sauer. They have served as an inspiration as I pursue my education and career. Their support throughout my education has blessed me with opportunities I otherwise would not have the luxury to experience.

ACKNOWLEDGMENTS

I would like to express my gratitude towards Dr. Guangming Xing, my first reader. I undertook this project under his suggestion, and only through his supervision and guidance was this project able to be completed.

VITA

EDUCATION

Western Kentucky University, Bowling Green, KY — May 2019
B.S. in Computer Science
Mahurin Honors College Graduate

Henderson County High School, Henderson, KY — May 2015

PROFESSIONAL EXPERIENCE

Gatton Academy, WKU — Sept. 2016 – May 2019
Computer Science Tutor

IT Department, Methodist Hospital — Apr 2015 – Jul 2015
Information Systems Technician

Customer Service Center, nGenx — Jul 2014 – Nov 2014
Customer Support Technician

AWARDS, SCHOLARSHIPS & HONORS

WKU President's List — 2015-2019
WKU 1906 Founders Scholarship — 2015-2019
AP Scholar with Honor — 2014, 2015

INTERNATIONAL EXPERIENCE

KIIS Study Abroad Program, Nara, Japan — June 2018
Hanyang International Summer School, Seoul, South Korea — July 2017

PRESENTATIONS

Sauer, R. (2019, March). *Containerized Scalable Security Training Environment.* Demonstration at the WKU Student Research Conference. Bowling Green, KY.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 BACKGROUND

1.1 Existing Training Methods

The primary issue with hands-on security training is that the training itself tends to break the environment in which the training takes place. Database information becomes altered or destroyed, and files get displaced or deleted. Furthermore, security training methods can't be practiced on websites or applications that do not belong to the tester, lest they become subject to legal ramifications. Therefore, it is necessary to have an environment that is rebuildable with ease, safe to break, and legal to tamper with.

Currently, to practice hands-on security vulnerability training, users must visit sites that provide a tutorial, such as Hacksplaining.com [1], or download a program that can create a virtual environment to launch a fully functioning website to train on, such as bWAPP [2]. Websites that offer vulnerabilities typically only offer a very limited selection of vulnerabilities to practice. They cannot be expanded to include more material, and their availability is dependent upon the owner, and not a guarantee. Websites may also only provide a simulated approach, where all the interactions are scripted, rather than having the ability to interact with a real system. The advantage websites do offer, however, is a centralized approach where all users connect to one

server. Downloadable programs, on the other hand, can offer more training exercises initially. If the programs offer the source files for deploying the environment, rather than a closed source program, then they are easily expandable and customizable. The disadvantage is that there is added overhead on the user's end of installation and maintenance as well as having sufficient hardware to manage and run the application.

There are several issues for schools and organizations wishing to have their students or employees use these solutions. Keeping a catalog of websites that offer training exercises can be tedious and the websites will not share a similar style or interface. This approach quickly becomes inefficient, especially if a website goes offline or makes changes to their content. Downloadable programs quickly become cumbersome to install, debug, and maintain on each individual setup. The more users an organization must manage, the more resources must be devoted to maintaining the multitude and variety of setups and configurations on a per-user basis. This project seeks to remedy the disadvantages of each approach while also combining the advantages to form a more robust, streamlined solution.

1.2 Proposed Solution

This project provides such a solution in the form of a server hosted environment. The advantages of this approach include a centralized application, where installation and maintenance are required for only one machine. The environment is customizable, allowing administrators to customize the available exercises to their needs and to their hosting specifications. The exercises will contain a genuine server environment that a user can attempt to exploit and damage, without affecting the stability of the overall application. This is achieved by running the exercises in isolated, secure containers that simulate a real machine in a sandbox environment, similar to a virtual machine. The desired environment, availability, and comprehensiveness of the application can be fully customized to any institution's preferences and needs.

Through any modern browser, users can log in to the application to create, destroy, and build the containers to test security vulnerabilities. Users will be able to create multiple containers under their name and have access to each at any time. A container is only accessible by the user who created it. One user's actions inside one of their containers will not interfere with another user's container or threaten the stability of the system. The application is scalable and can handle a few or many users concurrently. This allows it to be hosted on low or high-performance servers. The biggest benefit is that administrators will be able to add containers that expose different types of vulnerabilities and easily manage existing containers. There is nothing to install or maintain on the

user's device, as the server handles everything. The server hosted environment provides a centralized website approach with the feel and capabilities of a downloadable program.

Through this project, users will gain a deeper understanding of how security vulnerabilities are exploited and how they are performed. Such knowledge will prove useful in mitigating security vulnerabilities and learning proper programming techniques to avoid vulnerabilities.

2 TECHNOLOGY & TOOLS

2.1 Docker

Docker is a program that runs software packages and operating systems inside of containers. Each container functions independently from another. This prevents one crashing container from crashing the entire application. Every service that runs on this application runs inside a container using Docker.

Docker offers several advantages compared to running an application natively on the system or inside a virtual machine. Containers provide portability, so the application can be run on most systems without any extra build steps or procedures. Regardless of the development environment, if Docker officially supports the production environment the application will run the same. Docker also provides isolation capabilities. Containers do not have direct access to the host's system which helps prevent unauthorized access and modifications to the server [3]. The isolation creates a secure and safe environment for running web-accessible applications. Lastly, each container shares the OS kernel rather than replicating it, allowing for a much smaller footprint and faster load times [4].

2.2 Apache

Apache is an open-source HTTP server that provides secure and efficient HTTP services that follow HTTP standards [5]. It is a widely used and tested server that provides the necessary functions to complete the required routing between the containers.

When communicating with the websites on the application, the requests get sent to the Apache server which serves the webpages. PHP can be installed as a module to Apache which allows connections to the database. This is the current implementation to communicate with the MySQL database.

2.3 MySQL

MySQL is a relational database management system that is very fast, reliable, and scalable [6]. SQL stands for Structured Query Language and is a standard database language. The relational aspect means that data is separated and stored into separate tables rather than in one large table. Tables can also be referred to as relations. MySQL is used both for the database of the main application as well as the database inside the SQL injection container that is provided with the application.

Tables are uniquely identified by their names, and therefore must be unique. Tables consist of rows and columns. A column is a set of data values that are of the same type. Columns can also be referred to as attributes. Rows contain the structured data that populate the columns, following the data type specified for each column. Rows can also be referred to as records. To access data from a table, queries must be composed with proper formatting. The query is then executed by the database, returning a list of data that matched the query.

2.4 Alpine

Alpine is a lightweight Linux distribution for users who appreciate security, simplicity, and resource efficiency [7]. With its small size, it allows containers to be efficient and highly scalable. The simplicity helps maintain a barebones implementation without unnecessary and potentially vulnerable features, which lends to its security. All the containers provided use Alpine as the base OS image except for the MySQL container, which is Debian based.

2.5 PhpMyAdmin

PhpMyAdmin is a software tool to handle the administration of a MySQL database [8]. This allows administrators to easily manage databases through a GUI and quickly update and modify databases and their tables. The interface is not accessible to the regular users of the application, only to administrators. PhpMyAdmin was used to create the database used for the application and to dump the database for installation on other servers (see Appendix C).

3 SYSTEM DESIGN

Three containers are used to host the initial webpage that users use to log in and interact with the application. These containers are Apache, MySQL, and phpMyAdmin. The containers are all connected to the bridge network, the default network driver for docker. PhpMyAdmin is only used by administrators. The Apache container is part of another network, container-net, that contains all the containers launched by the users.

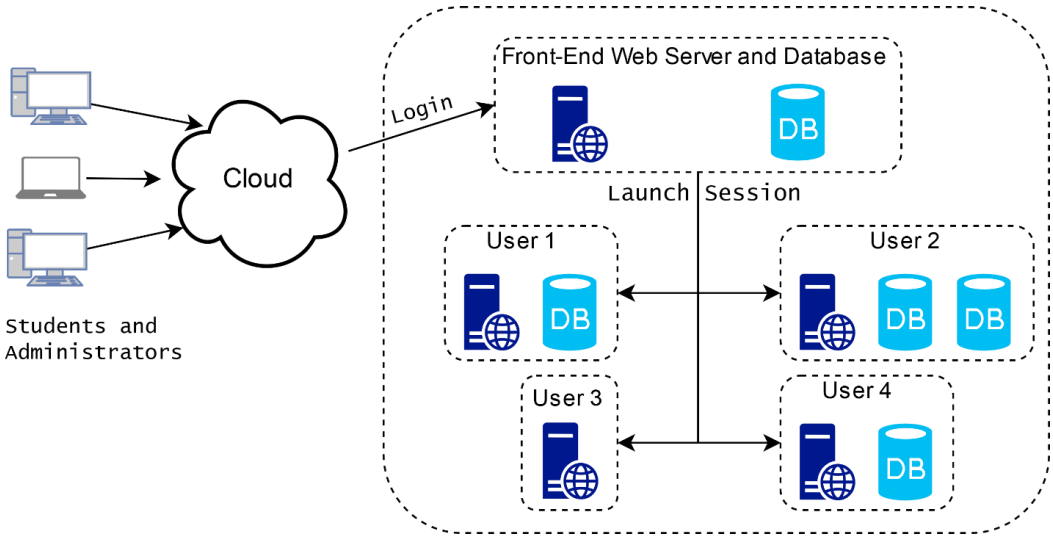


Figure 3.1. Logical diagram showing the layout and breakdown of the application.

The only way to access these containers will be through the Apache container. A direct connection will not be allowed to prevent unauthorized access and tampering. Attempting to access containers directly results in an error 403, meaning access is

forbidden. Figure 3.1 shows the logical layout of the project and how each component is connected.

The database for the application is named “security”. It is not a requirement that the database must have a specific name. However, changing the database name will require editing the connection files to allow Apache and phpMyAdmin to connect to the database. This database consists of two tables, Account and Container. Each account can have zero or many containers. Figure 3.2 is an ER diagram displaying the relationship between the tables and their fields. More information regarding the details of each table can be found in Appendix C.

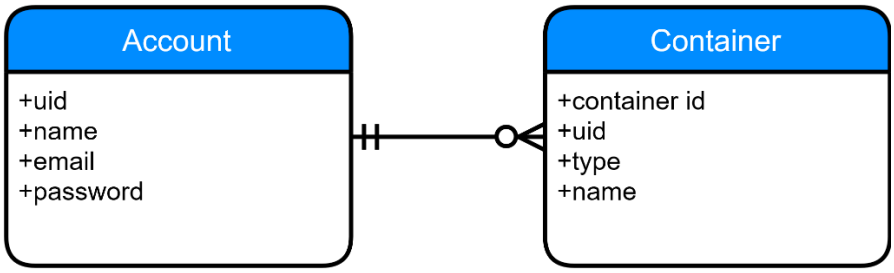


Figure 3.2. Entity relationship diagram of the security database.

Figure 3.3 is a use case diagram to demonstrate the interactions of users on the application. On the left is the actors on the system. Actors are entities, in this case, a specific user, that can perform an action on the application. In the center are the use cases, which consist of actions, services, and functions available on the system. The lines between an actor and a use case demonstrate the abilities and actions an actor can perform. On the right are the system components that the use cases effect. The lines

between the use cases and the system components demonstrate the system components that each use case effects. All actions and communication performed on the application pass through the Apache container.

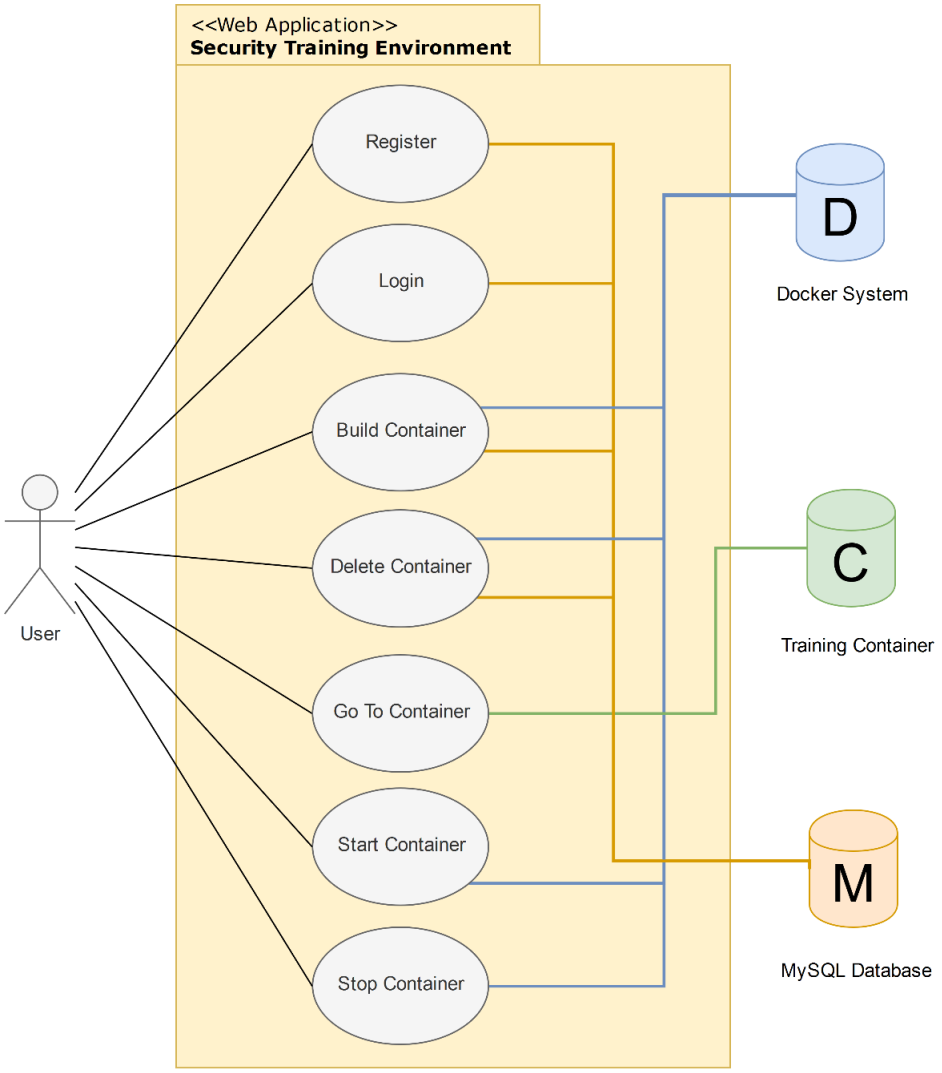


Figure 3.3. Use case diagram demonstrating the actions that a user can perform and which part of the system each action interacts with.

3.1 Installation

To install the application, it is recommended to use an operating system officially supported by Docker. The platforms include Windows, Mac, CentOS, Debian and Fedora, and Ubuntu [9]. While it is possible to host the application on an unsupported system, it is not guaranteed to be fully functional. The application fits inside of one folder and comes with an automatic installation script, "build.sh" (see Appendix E). You must have docker already installed on the system before running this script. If you are running installing the application on a Linux machine, the easiest way to run and monitor the script is to open a terminal, navigate to the folder, and run these two commands:

```
sudo chmod +x build.sh
sudo ./build.sh
```

The first command allows the script file to be executed, and the second command executes the script. Once the script has started, it will complete within several minutes, depending on the performance of the system. Once complete, the server will already have the three primary containers running and available via the web browser. The Apache container is exposed on the default port 80. The MySQL container is also exposed on its default port, 3306. The phpMyAdmin container is exposed on port 8080 since port 80 is already reserved for the apache container.

If a user is attempting to log into the website from an external device, then the website will be available on the domain that is configured for the server. If a user is attempting to log into the website from the server or machine it is hosted on, then the website can be accessed via the localhost address.

3.2 System Navigation

The following sections will use the “localhost” hostname to refer to the different URLs each page is found under. If a user is using an external device to connect to the application, replace “localhost” with the domain address configured for the server.

The first three pages of the website, “Home”, “Login”, and “Create Account”, are under the root path of the website, “localhost”. The “Student Dashboard” page is located at the URL “localhost/student”. When connecting to containers, the URL becomes “localhost/student/container/”. All containers redirect to this URL despite being a different type or belonging to different users. These are the paths defined by the application. Any other path appended to “localhost/student/container/”, is dependent upon the website hosted within each of the containers.

3.2.1 Home Page

The first page you reach is a simple landing page with the name of the website and a button for the user to log in. The URL for this page is simply “localhost”. Clicking the button will take you to the “Login” page.

Figure 3.4. Header and button displayed on the Home page.

3.2.2 Login Page

This page is where the user can sign in to their accounts with their email address and password. The email address field provides email validation that forces a properly formed email address to be supplied. The password field forces the minimum password length before being accepted. Before a request for signing in is sent to the server, these forms must be properly filled in.

Error messages are displayed above the form to provide the user with helpful information. Such errors include “Incorrect email/password combination”, and “Could not establish connection to server”. The first error message is straightforward. The second error message means that the connection to the database could not be established. This error will require an administrator to resolve the problem.

Below the form are two clickable links, “Create an account”, and “Main Menu”. “Create an account” will direct you to the “Create Account” page where new users can sign up to use the website. “Main Menu” will take you back to the home page. Upon successfully signing in, the user will be directed to the “Student Dashboard” page.

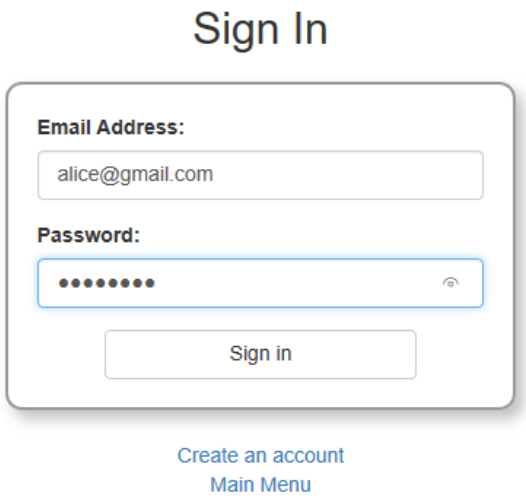


Figure 3.5. The form displayed on the Login page. Sample information is shown in the fields. The form is accompanied with a header above and links to other pages below.

1.1.1 Create Account Page

This page allows new users to sign up to the website. All information listed in the form on this page is required to sign up. The name of the user does not have to be unique. The email address must be unique as it is the primary identifier for an account. Therefore, a user may not create multiple accounts with the same email address. The email address field provides email validation. The password field has a restriction that it must be 6

characters long. This prevents short, easy to guess passwords from being used and increases the security of the application.

Popup hints will help in correcting the form before submitting, such as fields being left blank, passwords not matching, or an email incorrectly formed address. There are two links found below the form. The “Sign in to Account” link will take you the “Login” page. The “Main Menu” link will take you to the “Home” page. Upon successfully creating an account, the user will be directed to the “Student Dashboard” page.

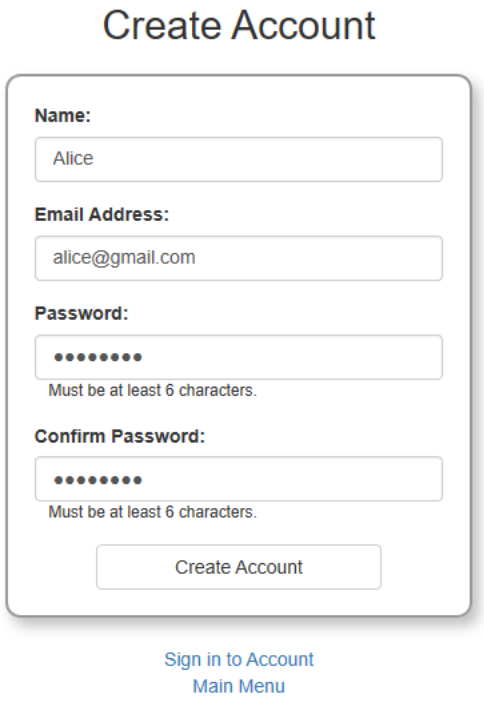


Figure 3.6. The form displayed on the Create Account page. Sample information is shown in the fields.

3.2.3 Student Dashboard Page

This page is where users can interact with containers. The available actions users can perform are:

- Building a container
- Starting a container
- Stopping a container
- Destroying a container
- Redirecting to a container's website

A separate panel is created for each different type of container. There is only one type of container provided at this point. The provided container is vulnerable to SQL injection. The title of the panel has been named appropriately to reflect this. A newly created user will not have any containers associated with their account, and therefore "No Containers" is displayed inside the respective panel as a message for the user. Once containers have been created, the message will be replaced with a drop-down box where the user may select the container to apply actions on.

Along the top of the navbar are the site name, the "Dashboard" tab, a welcome message for the currently signed in user, and a "Log Out" button. Attempting to navigate back to the "Home", "Login", or "Create Account" pages will automatically redirect the

user back to this page. To get back to any of the mentioned pages, the user must first log out by clicking the “Log Out” button to properly destroy the current user session.

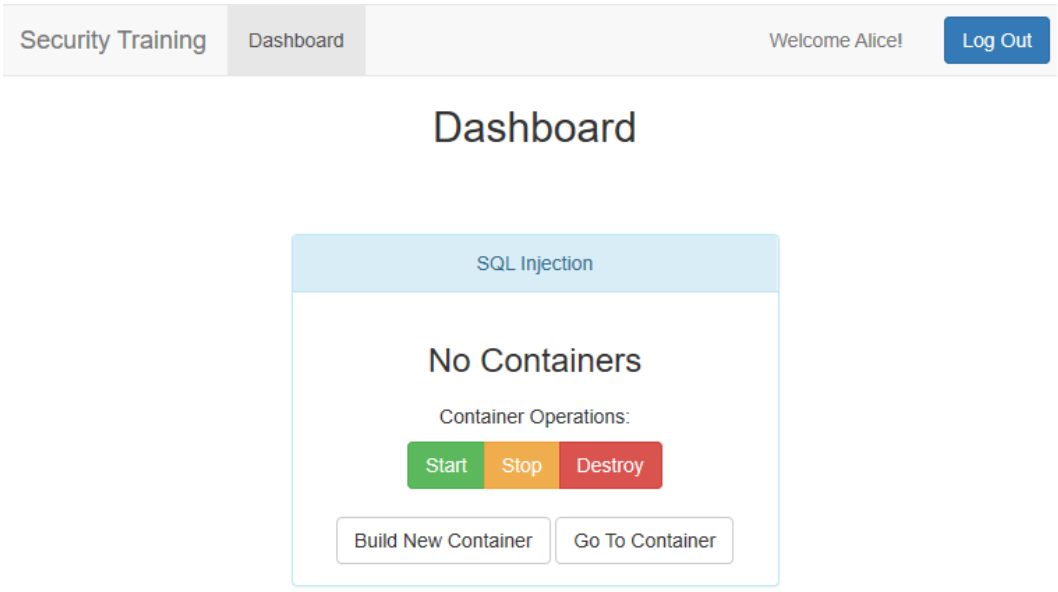


Figure 3.7. Student Dashboard page. The welcome message changes to each user’s name and the panel changes depending on whether the user has created containers.

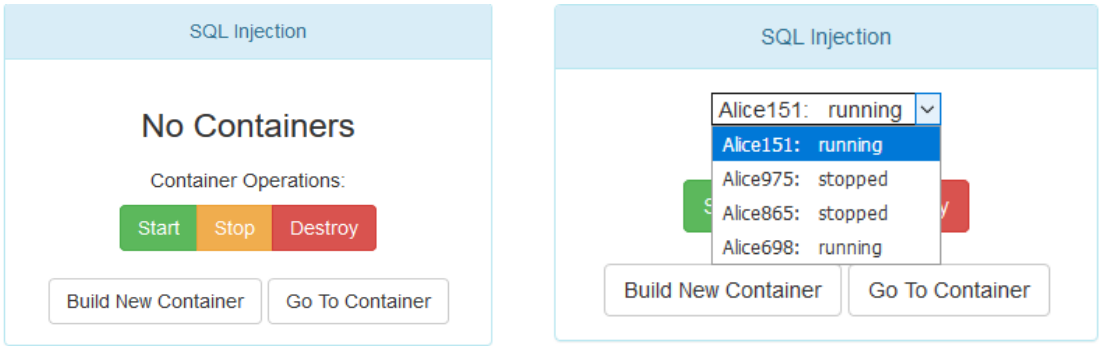
3.2.4 Container Page

This page changes depending on the container you are connecting to. Different types of containers can display different websites, each with their own structure and vulnerabilities. Furthermore, there are no links on these pages that can take you back to the “Student Dashboard” page. This is intended, as the container’s website is meant to be an isolated system unaware of the application we are using to host it. To get back to the “Student Dashboard” page, the URL must be manually edited to remove any path after

"student". For instance, the container page with a URL like "localhost/student/container/…" must be edited to "localhost/student/". Once back to the "Student Dashboard" page, the user must use the "Go To Container" button to return to the container. Clicking the browser back button or manually typing "container" back into the URL will not let you return to the container's website.

3.3 Manipulating Containers

A new user starts with no containers associated with their account. At this point, no buttons within the panel will function, other than the "Build New Container" button. Clicking on one of the other buttons will produce no messages and cause no errors. After a container is created the other buttons will function as normal. Figure 3.8 demonstrates how the panel looks with no containers and how it looks with several containers.



(a) (b)

Figure 3.8. Two container panels showing differences of a user (a) without containers and (b) with containers.

3.3.1 Building a Container

Clicking the “Build New Container” button will generate a new container. Once the container is created, the page will automatically refresh. The drop-down box in the panel will have appeared and will display the newly created container as an option. Generating more containers will add them to the drop-down box in no particular order. The options are displayed in the same order as their retrieval from the database. This typically results in the most recently created container appearing at the bottom of the list. There is no limit to the number of containers a user may have either per container type or overall across all container types.

Each container is named is a concatenation of the user’s name and a random number. The names are not required to be unique, though the possibility of having duplicate names is very low. There are no side effects or issues should two containers have the same name. Alongside the name of each container inside the drop-down box is the status of the container. There are only two statuses that can be displayed: “running” and “stopped”.

3.3.2 Starting a Container

A container that is stopped can be started by selecting the container from the drop-down box and then clicking on the “Start” button. The page will load for several seconds after clicking on the “Start” button. The browser is waiting for confirmation from the server on whether the operation was successful. Once a response has been received, or

after a timeout, the page will refresh and display a message informing the user of the success or failure of the operation. This operation is the longest of all the available operations.

Upon successful completion, the status of the container is updated to show that is now running. Attempting to start a container that is already running has no negative consequences and will not cause errors. Doing so will simply cause a message to be displayed informing the user that the selected container is already running.

3.3.3 Stopping a Container

To stop a running container, select the container from the drop-down box and then click the “Stop” button. The page will load for several seconds after clicking the “Stop” button. The page will refresh once the browser receives confirmation from the server on success or failure of the operation.

Upon successful completion, the status of the container is updated to show that is now stopped. Attempting to stop a container that is already stopped has no negative consequences and will not cause errors. Doing so will simply cause a message to be displayed informing the user that the selected container is already stopped.

3.3.4 Deleting a Container

To delete a container, select the container from the drop-down box and then click the "Destroy" button. The page will load for several seconds after clicking the "Destroy" button. The page will refresh once the browser receives confirmation from the server on the success or failure of the operation.

Upon successful completion of the operation, the container is removed from and no longer displayed within the drop-down box. The container will have been deleted both from the database and from the docker system. Docker requires that a container is stopped before allowing it to be deleted. However, the "Destroy" button can delete a container that is either stopped or currently running. If the container is stopped, the operation proceeds as normal by performing the delete operation on the docker system and database. If the selected container is running, the stop function is first called on the container. Once the container has successfully been stopped, only then will the delete operation be performed on the docker system and the database.

3.3.5 Navigating to a Container

To navigate to a container, click on the "Go To Container" button. The page will redirect to the "localhost/student/container/" URL. Under normal circumstances, there should be no difference in responsiveness between accessing the standard pages on the application versus accessing the pages within a container. There are no buttons, links, or

other features inside of a container's website that will assist the user in returning to the "Student Dashboard" page.

A container can only be accessed if the container is running. Attempting to access a stopped container will produce an error message informing the user that the container is currently stopped and that the URL for the website cannot be fetched.

3.4 Performing a SQL Injection Attack

This section will walk through how to do a SQL injection attack on the container that is provided with the application. A SQL injection attack allows SQL commands from the user to be sent and executed by the database on the backend. This potentially allows any user to gain access to sensitive information, delete information from the database, or perform privilege escalation.

First, build a SQL injection container, start the container, and then navigate to that container's website by clicking on the "Go To Container" button. You will be redirected to the container's home page as seen in Figure 3.9. This website is called pollinator and allows you to create polls that other users can vote on. Currently, there are three polls on the website. We are going to delete the database through SQL injection, which is also named pollinator.

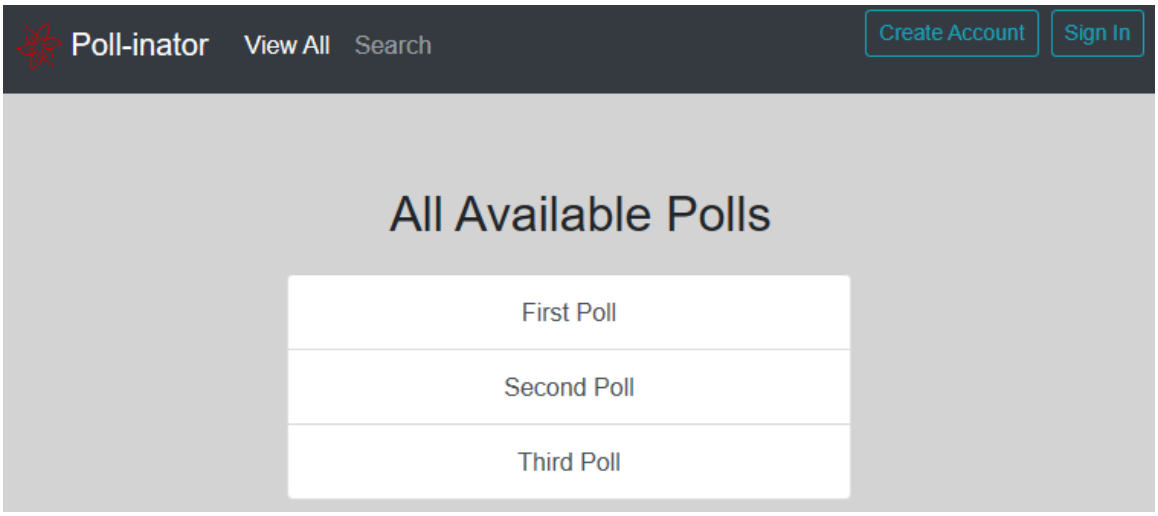


Figure 3.9. The home page of the SQL injection container website.

The vulnerable section of this website is the “Login” page, specifically the login form. Click the “Sign In” button on the top right inside the navbar to be directed to that page. Once there, instead of attempting to log in with an actual user, we will instead insert a SQL command for the username and 6 random characters in the password field. The characters in the password field are only necessary as the website requires a password entry before sending the information to the database. In Figure 3.10, the username field shows the SQL command to be typed in. Once you have filled both fields, attempt to log in via the “Sign In” button.

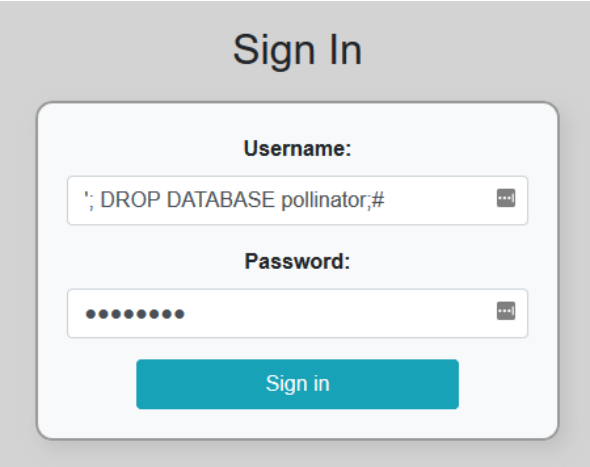


Figure 3.10. The form on the Login page of the pollinator website. A SQL command is in the username field, with random characters in the password field.

The page will refresh, and you will be presented with an error message as shown in Figure 3.11. The database for the website has now been dropped. When we attempt to return to the other pages on the website we will now be presented with errors.

The single quote before the first semicolon finishes the first SQL statement that fetches your username from the database to then compare the password submitted versus the password stored in the database. By adding a semicolon, we can add an additional SQL statement that deletes the database. The hashtag symbol removes extra characters that are leftover to create a properly formatted SQL string.

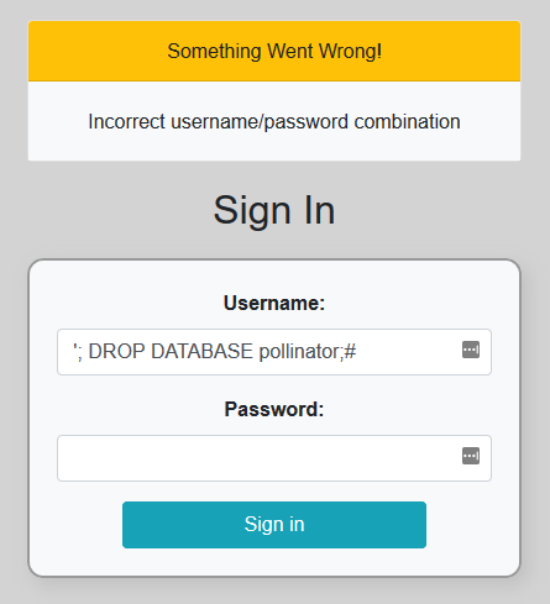


Figure 3.11. Error message received after performing SQL injection attack.

Now when attempting to go back to the “Home” page of the website, instead of seeing the three polls that were there originally, we are presented with an error message as shown in Figure 3.11. The website is now broken. Every page on the website will display this message. The user may return to the “Student Dashboard” page to delete this container and create a new container to test out other SQL commands. Other options of attacks in this category include modifying data in the database, retrieving a complete copy of all the information in the database, gaining access to other people’s accounts, or escalating privileges of an account.

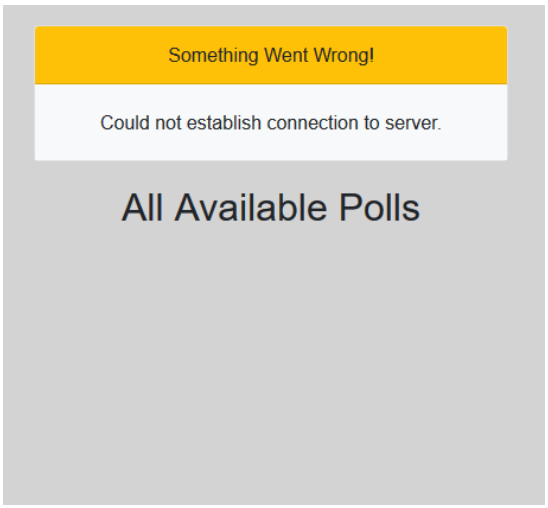


Figure 3.12. The error message that will be displayed on every page of the website once the database has been deleted.

4 CONCLUSION

This project provides a user-friendly, maintainable, and expandable server application that users can interact with on any internet capable device, in the form of a server hosted environment. The portability and scalability of the application allow it to be hosted on varying sets of equipment depending on the available budget. The customizability of available containers gives administrators the ability to tailor the application to a specific set of requirements and provide ample training exercises for their users. By using this project, users gain a deeper understanding of security vulnerabilities which can be put forth to mitigate security vulnerabilities in the future.

5 FUTURE WORK

In the future, more pre-built containers with a varying set of vulnerabilities can be provided with the initial application. Examples include cross-site scripting (XSS), LDAP and CLRF injection, XML external entity (XXE) processing among many others.

An administrator login could be added to allow remote management of the system providing a quick rundown of the server status, running containers, and other useful information. It could also be possible for administrators to upload new containers to be built and distributed to the user's automatically rather than the current manual process.

The current installation provides some customization to the project in terms of the database name, the password used, and the name of the container network. However, these fields do not propagate changes universally and therefore a more in-depth process is required to make the necessary changes. A more robust build script could solve these issues and provide an easier process for customization.

REFERENCES

[1] "Security Training for Developers," [Online]. Available: https://www.hacksplaining.com/. [Accessed 4 April 2019].

[2] "Buggy Web Application," [Online]. Available: http://www.itsecgames.com/. [Accessed 6 April 2019].

[3] "Docker Security," [Online]. Available: https://docs.docker.com/engine/security/security/. [Accessed 8 April 2019].

[4] "What is a Container?," [Online]. Available: https://www.docker.com/resources/what-container. [Accessed 12 March 2019].

[5] "Apache HTTP Server Project," [Online]. Available: https://httpd.apache.org/. [Accessed 3 April 2019].

[6] "What is MySQL?," [Online]. Available: https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html. [Accessed 21 March 2019].

[7] "About Alpine Linux," [Online]. Available: https://alpinelinux.org/about/. [Accessed 8 December 2018].

[8] "Bringing MySQL to the Web," [Online]. Available: https://www.phpmyadmin.net/. [Accessed 9 March 2019].

[9] "About Docker CE," [Online]. Available: https://docs.docker.com/install/. [Accessed 28 October 2018].

APPENDIX A: APACHE CONTAINER SOURCE CODE

The files listed are used by the Apache container to set up the environment and interact with the other containers. The Dockerfile is a script that docker uses to automatically build an image of a container. The five shell files, extensions ending in ".sh", are used to send commands to the Docker system. The "Student Dashboard" page is defined under A.7. The functions for the buttons on the "Student Dashboard" page are defined in A.8. The code that retrieves the container's website and returns it to the user is defined in A.9.

A.1 dockerfile

```
FROM ulsmith/alpine-apache-php7
RUN apk add docker shadow apache2-proxy
COPY /src /app/public
COPY httpd.conf /etc/apache2/httpd.conf
RUN chown apache:apache -R /app/public/student/container/
RUN chmod 755 -R /app/public/student/shell/
```

A.2 create.sh

```
#!/bin/ash
docker create --name $1 --network container-net container/$2
```

A.3 destroy.sh

```
#!/bin/ash
```

```
docker rm $1
```

A.4 start.sh

```
#!/bin/ash
docker start $1
docker exec $1 sh -c "rc-service mysql start"
```

A.5 status.sh

```
#!/bin/ash
docker inspect -f '{{.State.Running}}' $1
```

A.6 stop.sh

```
#!/bin/ash
docker stop -t 0 $1
```

A.7 student/index.php

```
<?php
include("session.php");
include("../connection.php");
$active = array("active");

include("actions.php");
include("../gen_html.php");
genHTML("Dashboard");
?>
<link rel="stylesheet" type="text/css" href="../css/content.css">
</head>

<body>
    <?php include("navbar.php"); ?>
    <h2>Dashboard</h2>
    <div class="header-container">
        <?php display_error();
        display_success(); ?>
    </div>

    <div class="panel panel-info form-wrapper">
        <div class="panel-heading">SQL Injection</div>
```

```
        <div class="panel-body">
            <form id="cont-sql-inj" action="./index.php" method="post">
                <input type="hidden" name='type' value='sql-inj'/>
                <?php listContainers("sql-inj");?>
            </form>
            <div class="row">
                <div class="col-sm">
                    <h5>Container Operations:</h5>
                    <div class="btn-group" role="group" aria-label="...">
                        <input form="cont-sql-inj" id="start" name="cont-
op" type="submit" value="Start" class="btn btn-success"/>
                        <input form="cont-sql-inj" id="stop" name="cont-
op" type="submit" value="Stop" class="btn btn-warning"/>
                        <input form="cont-sql-inj" id="destroy"
name="cont-op" type="submit" value="Destroy" class="btn btn-danger"/>
                    </div>
                </div>
            </div>

            <br>
            <div class="row">

                <input form="cont-sql-inj" type="submit" value="Build New
Container" name="build" class="btn btn-default"/>
                <input form="cont-sql-inj" type="submit" value="Go To
Container" name="goto" class="btn btn-default"/>
            </div>
        </div>
    </div>
</body>
</html>
```

A.8 student/actions.php

```
<?php
if (isset($_POST['cont-op']) && isset($_POST['exists'])) {
    switch ($_POST['cont-op']) {
        case 'Start':
            startContainer($_POST['container']);
            break;
```

```
        case 'Stop':
            stopContainer($_POST['container']);
            break;
        case 'Destroy':
            deleteContainer();
            break;
    }
}else if(isset($_POST['build'])){
    buildContainer();
}
else if(isset($_POST['goto']) && isset($_POST['exists'])){
    global $errors;
    $status = getStatus($_POST['container']);

    if($status == "running"){
        $_SESSION['container'] = $_POST['container'];
        header("Location: container/");
        die();
    }else{
        $errors[] = "Container Is Not Running. Can't Fetch URL.";
    }
}

function buildContainer(){
    global $db, $errors, $success;

    $uid = $_SESSION['uid'];
    $cont_id = uniqid('',false);
    $type = $_POST['type'];
    $name = substr(preg_replace("/[^a-zA-Z0-9]+/", "", $_SESSION['name']),
0, 6);
    $name = $name.mt_rand(1,1000);
    $containers = getContainers("all");

    while(!isUnique($containers, $cont_id)){
        $cont_id = uniqid('',false);
    }

    createContainer($cont_id, $type);
    if(count($errors) !== 0){
        return;
```

```
    }

    $stmt = $db->prepare("INSERT INTO container (container_id, name, type,
uid) VALUES(?,?,?,?)");
    $stmt->bind_param("ssss",$cont_id, $name, $type, $uid);

    if (!$stmt->execute()) {
        $errors[] = "Error Creating Container: DB Error";
    }

    $stmt->close();
}//End buildContainer

function deleteContainer(){
    global $db, $errors, $success;

    $uid = $_SESSION['uid'];
    $cont_id = uniqid('',false);
    $type = $_POST['type'];
    $name = substr(preg_replace("/[^a-zA-Z0-9]+/", "", $_SESSION['name']),
0, 6);
    $name = $name.mt_rand(1,1000);
    $containers = getContainers("all");

    $cont_id = $_POST['container'];
    destroyContainer($cont_id);
    if(count($errors) !== 0){
        return;
    }

    $stmt = $db->prepare("DELETE FROM container WHERE container_id=?");
    $stmt->bind_param("s",$cont_id);

    if (!$stmt->execute()) {
        $errors[] = "Error Deleting Container: DB Error";
    }

    $stmt->close();
}//End deleteContainer

function listContainers($type) {
```

```
    $containers = getContainers($type);
    if(count($containers) > 0){
        ?><select name="container" id="cont-list"><?php
        foreach ($containers as $con){
            $status;

            if(count($errors) > 0){
                $status = "err";
            }else{
                $status = getStatus($con['container_id']);
            }

            $spacing = ": ".str_repeat(" ",10-
strlen($con['name']));
            $name = $con['name'].$spacing.$status;
            ?><option
value=<?="{$con['container_id']}"?>><?="{$name}"?></option><?php
        }
        ?>
        </select>
        <input type="hidden" name="exists" value="true">
        <?php
    }else{
        ?>
        <h3>No Containers</h3>
        <?php
    }
}//End listContainers

function getContainers($type){
    global $db, $errors;
    $uid = $_SESSION['uid'];

    if($type == "all"){
        $stmt = $db->prepare("SELECT * FROM container WHERE uid=?");
        $stmt->bind_param("s", $uid);
    }else{
        $stmt = $db->prepare("SELECT * FROM container WHERE uid=? AND
type=?");
        $stmt->bind_param("ss", $uid, $type);
    }
```

```
    if (!$stmt->execute()) {
        $errors[] = "Error Retrieving Containers.";
        $stmt->close();
        return;
    }

    $result = $stmt->get_result();
    $stmt->close();

    return mysqli_fetch_all($result, MYSQLI_ASSOC);
}//End getContainers

function isUnique($array, $value){
    $unique = true;

    foreach($array as $e){
        if(in_array($value, $e)){
            $unique = false;
            break;
        }
    }

    return $unique;
}//End isUnique

function getUserByEmail($email){
    global $db;
    $stmt = $db->prepare("SELECT * FROM account WHERE email=?");
    $stmt->bind_param("s", $email);
    $stmt->execute();
    $result = $stmt->get_result();
    $stmt->close();

    return mysqli_fetch_assoc($result);
}//End getUserByEmail

function startContainer($cont_id) {
    global $errors, $success;

    $status = getStatus($cont_id);
```

```
    if($status == "running"){
        $errors[] = "Container Is Already Running.";
        return;
    }

    $output = shell_exec('su root -s /bin/sh -c
"/app/public/student/shell/start.sh '.$cont_id.'" 2>&1');
    if(strpos($output,"SUCCESS!") === false){
        $errors[] = "Container Not Started: Shell Error";
    }else{
        $success[] = "Container Started Successfully.";
    }
}//End startContainer

function stopContainer($cont_id) {
    global $errors, $success;

    $status = getStatus($cont_id);
    if($status == "stopped"){
        $errors[] = "Container Is Already Stopped.";
        return;
    }

    $output = exec('su root -s /bin/sh -c
"/app/public/student/shell/stop.sh '.$cont_id.'" 2>&1');
    if(strpos($output," ") !== false){
        $errors[] = "Container Not Stopped: Shell Error";
    }else{
        $success[] = "Container Stopped Successfully.";
    }
}//End stopContainer

function createContainer($cont_id, $type) {
    global $errors, $success;
    $output = exec('su root -s /bin/sh -c
"/app/public/student/shell/create.sh '.$cont_id.' '.$type.'" 2>&1');

    if(strpos($output," ") !== false){
        $errors[] = "Container Not Created: Shell Error";
    }else{
        $success[] = "Container Created Successfully.";
```

```
    }
}//End createContainer

function destroyContainer($cont_id) {
    global $errors, $success;

    $status = getStatus($cont_id);
    if($status == "running"){
        stopContainer($cont_id);
    }

    $output = exec('su root -s /bin/sh -c
"/app/public/student/shell/destroy.sh '.$cont_id.'" 2>&1');
    if(strpos($output," ") !== false){
        $errors[] = "Container Not Destroyed: Shell Error";
    }else{
        $success[] = "Container Destroyed Successfully.";
    }
}//End destroyContainer

function getStatus($cont_id){
    global $errors, $success;
    $output = shell_exec('su root -s /bin/sh -c
"/app/public/student/shell/status.sh '.$cont_id.'" 2>&1');

    if(strpos($output,"false") === false &&
strpos($output,"true")===false){
        $errors[] = "Container Status Error: Shell Error";
        return "err";
    }

    return (strpos($output,"false") === false )? "running" : "stopped";
}//End getStatus

function display_error() {
    global $errors;

    if (count($errors) > 0){
    ?>
        <div class="panel panel-warning form-wrapper">
            <div class="panel-heading">Something Went Wrong!</div>
```

```
            <div class="panel-body">
    <?php foreach ($errors as $error){ ?>
                <?=$error?><br>
        <?php } ?>
            </div>
        </div>
    <?php
    }
}//End display_error

function display_success() {
    global $errors, $success;

    if (count($success) > 0 && count($errors) == 0 ){
    ?>
        <div class="panel panel-success form-wrapper">
            <div class="panel-heading">Success!</div>
            <div class="panel-body">
    <?php foreach ($success as $part){ ?>
                <?=$part?><br>
        <?php } ?>
            </div>
        </div>
    <?php
    }
}//End display_success
?>
```

A.9 student/container/index.php

```
<?php
session_start();
if(!isset($_SESSION['uid']) || !isset($_SESSION['container'])) {
    header("Location: /student");
    die();
}

$url = $_SESSION['container'].'/';
// create a new cURL resource
$ch = curl_init();
```

```php
// set URL and other appropriate options
if(isset($_GET['path'])){
    $url = $url.$_GET['path'];
    unset($_GET['path']);
}
if(count($_GET) > 0){
    $url = $url."?".http_build_query($_GET);
}

curl_setopt($ch, CURLOPT_URL,  $url);
curl_setopt ($ch, CURLOPT_FOLLOWLOCATION, 1);
curl_setopt ($ch, CURLOPT_AUTOREFERER , 1);
curl_setopt ($ch, CURLOPT_RETURNTRANSFER , 1);
curl_setopt ($ch, CURLOPT_COOKIEFILE, $_SESSION['container']);
curl_setopt ($ch, CURLOPT_COOKIEJAR, $_SESSION['container']);

if(strpos($url,'.css') != False){
    curl_setopt($ch, CURLOPT_HTTPHEADER,  array(
        'Content-Type: text/css; charset=UTF-8'
    ));
}
if(count($_POST) > 0){
    curl_setopt($ch, CURLOPT_POST, 1);
    curl_setopt($ch, CURLOPT_POSTFIELDS, http_build_query($_POST));
}

$page = curl_exec($ch);
$page = str_replace("<!DOCTYPE html>", "", $page);
curl_close($ch);
echo $page;
?>
```

APPENDIX B: TRAINING CONTAINER SOURCE CODE

The Dockerfile described here builds the image used to create the training container with the vulnerable websites.

B.1 dockerfile

```
FROM ulsmith/alpine-apache-php7
RUN apk add shadow mariadb mariadb-client openrc
COPY /src /app/public
RUN mysql_install_db --user=mysql --datadir='/var/lib/mysql'
RUN cp /usr/share/mariadb/mysql.server /etc/init.d/mysql
RUN chmod 755 /etc/init.d/mysql
RUN chmod 755 -R /app/public/
RUN rc-update add /etc/init.d/mysql default
COPY pollinator.sql /var/lib/mysql/pollinator.sql
```

APPENDIX C: MYSQL CONTAINER SOURCE CODE

C.1 is a table with details about the database relations. The index column lists restrictions for a given field. The primary key of a relation describes the fields used to distinguish entries from each. No two entries can have the same primary key or the same combination of primary keys if there are multiple. A foreign key defines a field that originates from the primary key of another relation. The “uid” field in the “Container” relation originates from the “uid” field in “Account”. C.2 is the SQL dump that can be imported into a database to automatically create the tables described in C.1.

C.1 Database Table

Table	Field	Index	Type	Length	Description
Account	uid	Primary Key	Character	23	Unique id identifying the account
	name		Variable Character	256	Name of the user.
	email	Unique Key	Variable Character	256	User’s email.
	password		Variable Character	256	Password stored after being hashed.
Container	container id	Primary Key Unique Key	Character	13	Auto-incrementing id for every project
	uid	Primary Key Foreign Key	Character	23	The unique id of the account that owns this container
	type		Variable Character	10	A string determining what type of container it is.
	name		Variable Character	10	Name of the container to be displayed to the user.

Table C.1. Table showing database structure and field settings.

C.2 security.sql

```
-- phpMyAdmin SQL Dump
```

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";

CREATE TABLE `account` (
  `uid` char(23) NOT NULL,
  `name` varchar(80) NOT NULL,
  `email` varchar(256) NOT NULL,
  `password` varchar(256) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;


CREATE TABLE `container` (
  `container_id` char(13) NOT NULL,
  `uid` char(23) NOT NULL,
  `type` varchar(10) NOT NULL,
  `name` varchar(10) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

ALTER TABLE `account`
  ADD PRIMARY KEY (`uid`),
  ADD UNIQUE KEY `email` (`email`);

ALTER TABLE `container`
  ADD PRIMARY KEY (`container_id`,`uid`),
  ADD UNIQUE KEY `container_id` (`container_id`),
  ADD KEY `uid` (`uid`);

ALTER TABLE `container`
  ADD CONSTRAINT `uid_ibfk_1` FOREIGN KEY (`uid`) REFERENCES `account`
(`uid`);
COMMIT;
```

APPENDIX D: PHPMYADMIN CONTAINER SOURCE CODE

The Dockerfile used to create the phpMyAdmin image.

D.1 dockerfile

```
FROM phpmyadmin/phpmyadmin:latest
```

APPENDIX E: AUTOMATED INSTALLATION SCRIPT

The script file that automatically builds the project. The variables described in the beginning can be changed to suit the needs of different environments. This does not propagate the changes everywhere, however. Changing the password or database variables, for example, would require changing the PHP files in Apache to connect to the database with the new values.

E.1 build.sh

```
#!/bin/bash
readonly PASSWORD="pass123"
readonly DATABASE="security"
readonly NETWORK="container-net"
readonly ROOT=$PWD

#Create Network
sudo docker network create $NETWORK

#Create MySql Container
#Gives mysql time to startup before performing MySql commands later
sudo docker run --name sec-mysql -d -e MYSQL_ROOT_PASSWORD=$PASSWORD -e
MYSQL_DATABASE=$DATABASE mysql:latest

#Create PHPMyAdmin Image
cd $ROOT/phpmyadmin
sudo docker build -t security/phpmyadmin .

#Create Apache Image
cd $ROOT/apache
sudo docker build -t security/apache .

#Create SQL-Inj Image
cd $ROOT/container
sudo docker build -t container/sql-inj .
```

```
#Copy Database SQL file to mysql for creation
cd $ROOT/mysql
sudo docker cp $DATABASE.sql sec-mysql:/var/lib/mysql/$DATABASE.sql

#Create Apache Container
cd $ROOT
sudo docker run --name sec-apache -dit -p 80:80 -v
/var/run/docker.sock:/var/run/docker.sock -v $ROOT/apache/src:/app/public/
--link sec-mysql:mysql security/apache
sudo docker network connect $NETWORK sec-apache

#Create phpMyAdmin Container
sudo docker run --name sec-myadmin -d -p 8080:80 -e PMA_HOST=mysql --link
sec-mysql:mysql security/phpmyadmin

#Create Temporary Container To Finish Building SQL-Inj
sudo docker run --name temp -dit --network $NETWORK container/sql-inj
sudo docker exec temp bash -c "rc-service mysql start;"
sudo docker exec temp bash -c "mysql --execute 'CREATE DATABASE
pollinator; USE pollinator; source /var/lib/mysql/pollinator.sql;'"
sudo docker exec temp bash -c "mysqladmin -u root password '$PASSWORD'"

#Grab IP address from apache container to allow access to containers only
from this IP address
IP="$(sudo docker inspect -f "{{with index .NetworkSettings.Networks
\"$NETWORK\"}}{{.IPAddress}}{{end}}" sec-apache)"
echo "${IP}"
sudo docker exec temp bash -c "sed -i 's/Require all granted/Require ip
$IP/' /etc/apache2/httpd.conf"
sudo docker commit -m "Initialized" temp container/sql-inj
sudo docker stop -t 0 temp
sudo docker rm temp

# Alter User fixes caching SHA2 error from phpmyadmin
until sudo docker exec -it sec-mysql mysql -u root -p$PASSWORD -e "ALTER
USER root IDENTIFIED WITH mysql_native_password BY '$PASSWORD'; CREATE
DATABASE IF NOT EXISTS $DATABASE;"; do
  echo MySql not yet running, retrying in 10 seconds...
  sleep 10
done
```

```
sudo docker exec sec-mysql mysql -u root -p$PASSWORD --execute "USE
$DATABASE; source /var/lib/mysql/$DATABASE.sql;"
sudo docker commit -m "Initial Container" sec-mysql security/mysql
```