10/15/2018

Cyber Crime and Security Enhanced Programming

Report

ID: 18821354

Christopher Hsu Chang
CURTIN UNIVERSITY

Table of Contents

Introduction:

The following report is for the Cyber Crime and Security Enhanced Programming assignment for 2018. The report will discuss about several security flaws and different vulnerabilities that exist in the web today categorised by OWASP. Provided, is a web application that is functional and able to buy/edit movies. The web application has been purposely coded to introduce these major security vulnerabilities. Throughout this report I shall demonstrate each type of vulnerability extensively, any possible exploits, and what you have to do to change/mitigate the vulnerability from happening.

Also in this report, is a brief overview on how the web application works, and what you have to do to set up the application on a fresh instance of the virtual machine provided.

Setting Up:

In order to set up the virtual environment from a fresh install, please do the following:

Setting Up the Website:

1. Download the latest copy of CCSEP_Assignment.ova (The second version that Lincoln Short gave to us) and boot it up.
2. Go to the /var/www/html directory and delete all of the files present inside that directory by typing these commands.
    - rm index.html
    - rm –rf js
    - rm –rf css

    (We are deleting all of these files because I have my own js and css with extra stuff other than bootstrap that we are going to use)
3. Now tab out of the terminal window and go to the assignment submission folder, copy all of the files that exist inside Website/html (This also including copying all of the directories (css, js, images, audio)) and copy them into the virtual machine's html directory. You can do this via a Shared Network using Samba if you are using Windows/OSX.
4. Now for testing purposes, open up your preferred browser (Chrome or Firefox) and go to the ip address 192.168.56.150 just to check you manage to find the homepage of the website. If you don't find the index.php page by typing in the ip address you have done steps 1, 2 or 3 incorrectly. If you have any problems trying to copy over files then try restart the server again.
5. Once you have completed steps 1-4 move onto setting up the database.

Now we have finished setting up the Website, we can begin to set up the database and insert the tables required to run this web application.

Setting Up the Database:

Courtesy of Jonathon Winter, for providing these files to set up the assignment. I modified the script so that you have to setup the website manually, but setting up the database is automated.

1. Copy the files “setup.sql, setupAssignment.sh, database.sql” from the Assignment Submission directory over to /var/www/ directory inside the virtual machine.
    - setup.sql – is the sql script that creates the database
    - database.sql – is the sql script that creates and populates all of the tables.
    - setupAssignment.sh – is a bash script that calls the two above .sql files
2. Run the command:
    - ./setupAssignment.sh
3. The database should be created now, for testing purposes log into the mysql server by typing
    - mysql –u ccsep –p
    - and the password is “ccsep_2018”
    - use assignment; (To go into the assignment database)
    - and if you type ‘SHOW Tables;’ you should be able to see four tables;
        - Movies
        - Purchases
        - Reviews
        - Users

When you see all of these tables, you have set up the database correctly.

Overview:

The Web Application is able to support all of these following operations:

1. Adding of users accounts
2. Addition of “funds” for users (RayCoins)
3. Purchasing of movies using funds
4. Reviewal of movies
5. Searching for movies
6. Addition and removal of movies by application administrator
7. Management (Edit fields) of user accounts by application administrator.

The application allows for two different types of users, (admin and not admin). The difference between them is that the admin has access to the admin panel in which they have access to change the password, funds and admin level of any given user/admin account. Accounts that have access to the admin panel, also have the ability to Create new movies and Remove existing movies.

Files:

- **addfunds.php**
  - Allows Users to add RayCoins to their account to purchase Movies
- **admin.php**
  - Provides template to dynamically load pages, userlisting.php and movielisting.php. This page was created for the php file include vulnerability, which gives a generic panel which allows admins to swap between the two php files.
- **database_con.php**
  - Provides functions that, other php files use in order to do things such as connect to the database and query the database.
- **header.php**
  - File that can be included in between <head> tags to provide any bootstrap libraries
- **index.php**
  - Home page of the website
- **login.php**
  - Login page of the website
- **modal_buttons.php**
  - Functions that provide templates (HTML) for the modal dialog boxes that the admin panel uses.
- **movie.php**
  - Link to each individual movie page, where accounts are allowed to purchase the movie
- **movielisting.php**

    - HTML and Functionality for Admin accounts to add/remove Movies
- **navbar.php**
    - Template that exists on every page, the provides an interface of navigation so users are able to go to different pages.
- **navbarButton.php**
    - Template for the Navigation Bar button show and hide the navigation bar that every page uses
- **profile.php**
    - Profile page for accounts, that is STILL UNDER CONSTRUCTION
- **purchase.php**
    - Page where users can browse for Movies and use the search bar to search for particular movies.
- **signout.php**
    - Contains code that deletes SESSION Variables when the logged in User chooses to sign out and then redirects the user back to the home page.
- **signup.php**
    - Signup page, for Users both Privileged and non-privileged to register User accounts
- **success.php**
    - Upon signing up, there is a welcome page, congratulating you on joining
- **userlisting.php**
    - HTML and Functionality for Admin accounts to edit/remove Users from the database.

Database:

By default, the database contains 100 entries of Movies, which can all be searched up in purchase.php. If you want to see all the entries in the table I recommend downloading mysql workbench and using that to load everything up.

The Purchases table is currently empty but it links the UserID with the MovieID so it can keep track of what Users have purchased what Movies.

The Reviews table is also currently empty, but it keeps track of what Users have been writing comments on certain movies. So each individual movie will have its own set of reviews.

Finally, the Users tables contains two entries, an admin user and a normal user.
The Username and Password for both users are:

- Username: admin
  Password: admin
- Username: user
- Password: user

How this thing works:

On upon, accessing the website you will be welcomed to the greeting of the homepage where there is nothing but a side navigation bar, prompting you to “Login” or “Signup”

Figure 1.1: index.php, The Home page for the Assignment

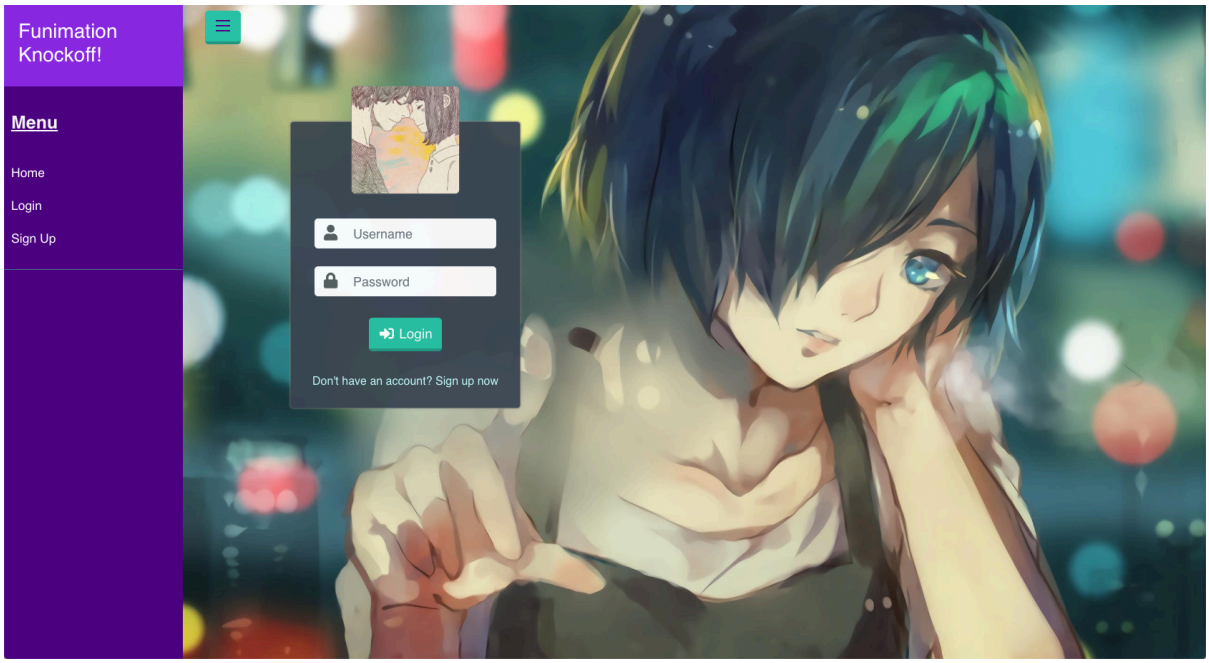


Figure 1.2: login.php, Login page so users can log in

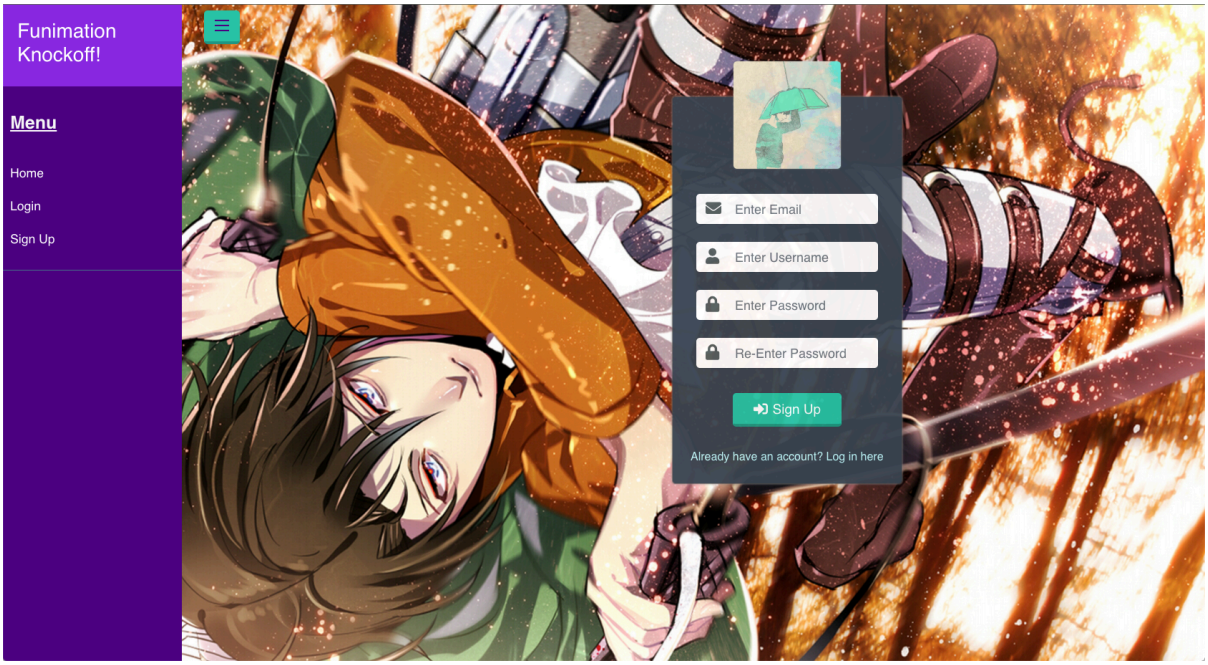


Figure 1.3: signup.php, Signup page so Users can create account

Upon logging in you will be redirected back to the home page, but the admin panel will provide extra options for you, dependent if you are a normal user or a privileged user. The difference is that Privileged users get access to an extra option called admin panel where they are allowed to Manage Users and Movies.

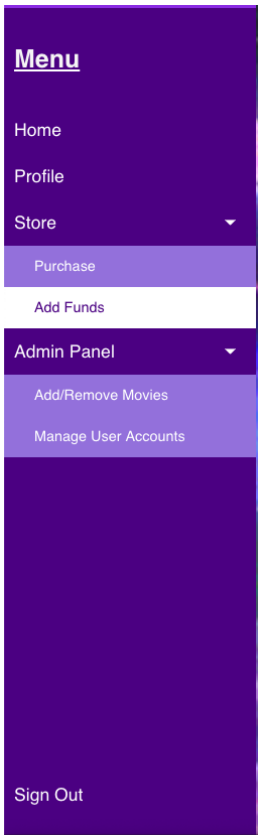


Figure 1.4: Admin Panel once Users have logged in

From then on, everything else is pretty self-explanatory. Purchase, just navigates you to a page to purchase movies, Add/Remove Movies navigates you to a page where you can add and delete movies etc.

Known Defects/Bugs:

There are not any known major bugs/defects in the web application that will cause any of the areas of where the marks/vulnerabilities are or with the general functionality of the web application but there are a few minor issues:

1. Inside the Manage User Accounts Page (userlisting.php), you cannot change funds of any User to be zero because I believe when trying to process this transaction, php treats zero as a Boolean (false) so it just ignores it completely.

Vulnerabilities:

1.XSS/Reflected:

Description:

Cross-Site Scripting (XSS) is a different type of code injection vulnerability that exists over the web, this vulnerability appears when the data provided by a web client is used immediately by server-side scripts to parse and display a page of results back to the user without properly sanitizing the original request. Thus the input coming from the HTTP client is being *REFLECTED* back to the user after being processed.

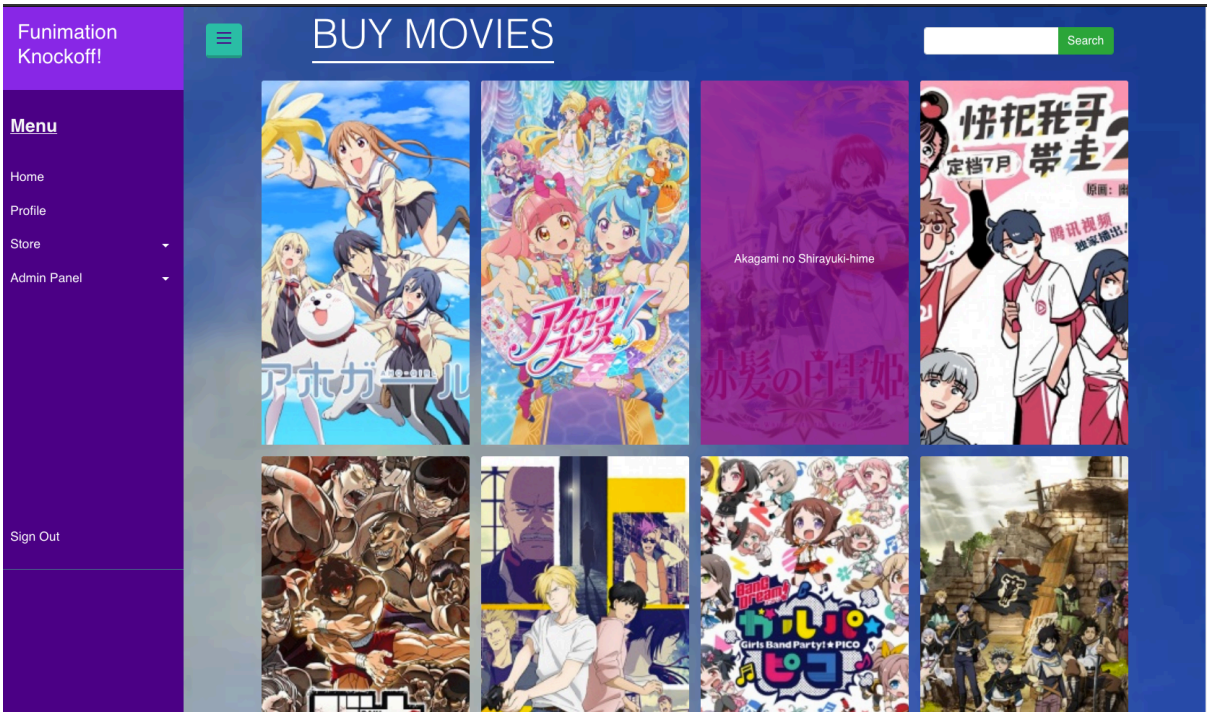
Possible Exploits:

There are several exploits that can be done when it comes to an XSS/Reflected type of attacks, this involves things such as Session Hijacking, where an attacker will use JavaScript to steal the current user's cookies as well as the Session Cookie. The reason why an attacker would want to steal this information is because, they won't need a Username or Password in order to log in as you. As long as they have your Session Cookie, they are able to log in, since it is like a ticket that identifies who you are.

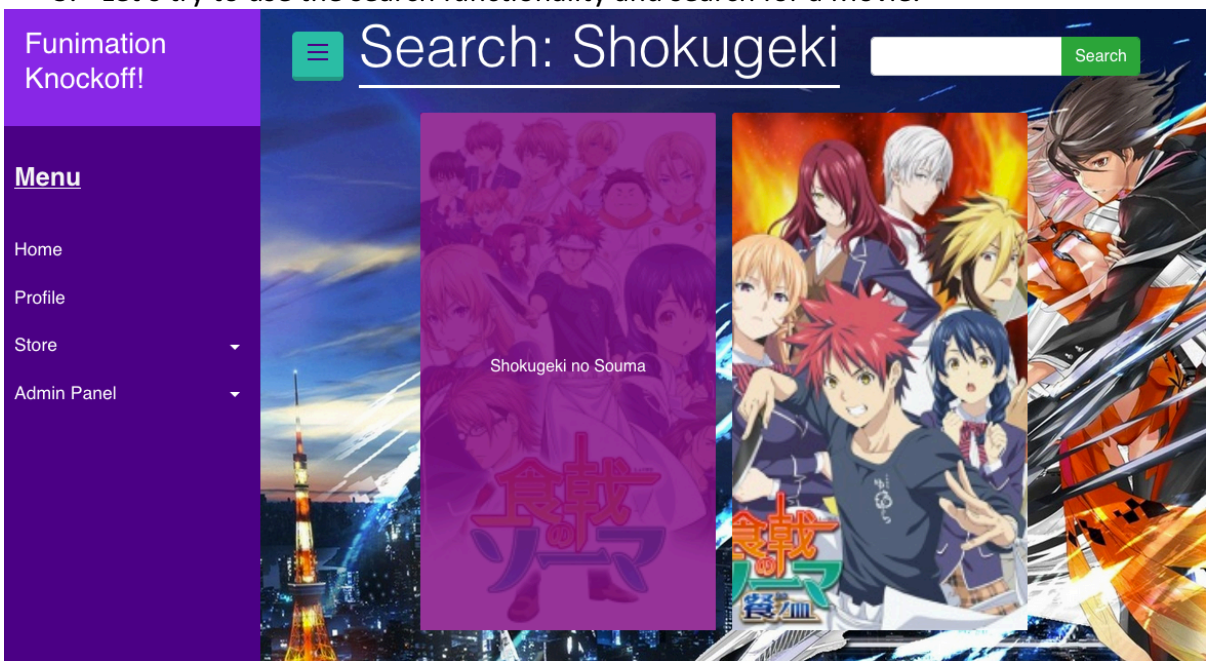
There are a lot of other malicious exploits that you can use with an XSS attack such as Redirect the victim to a malicious URL, phishing attacks and browser attacks.

Walkthrough:

1. Upon logging as any user navigate to the 'Purchase' link under the Store tab inside the Side Panel.
2. You will be presented with a page where you can scroll through or search for certain Movies inside the Database.

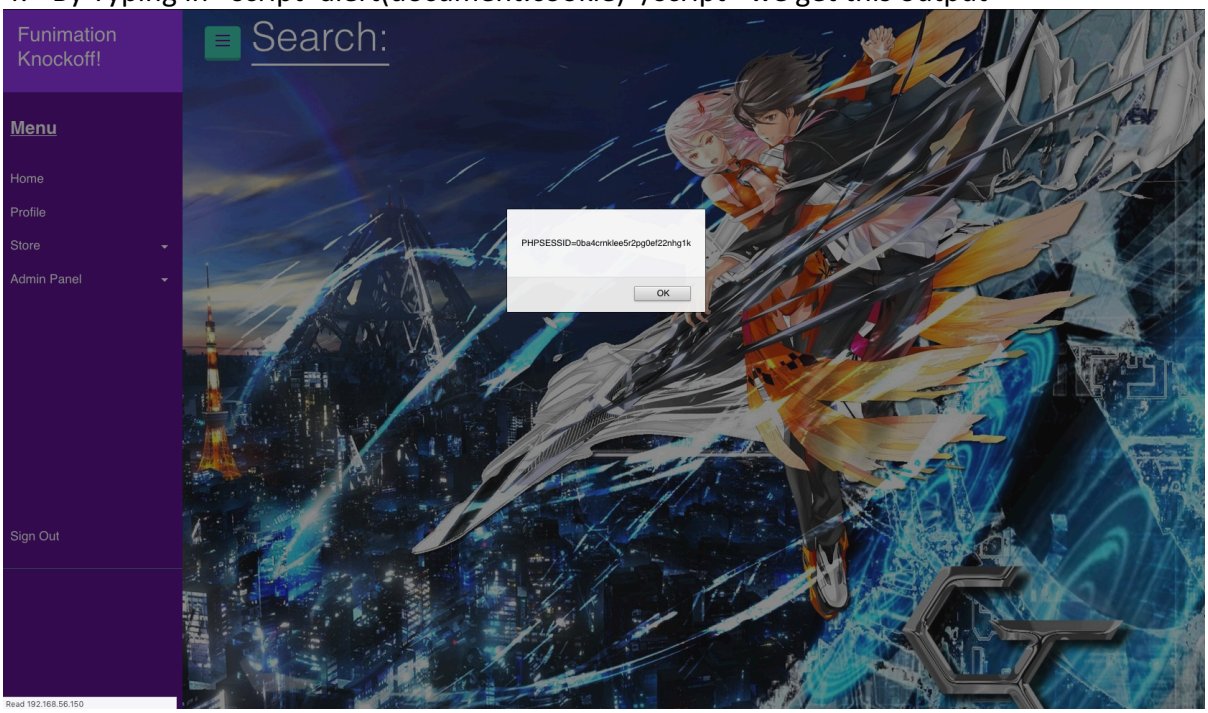


3. Let's try to use the search functionality and search for a Movie.



You can see here my search term was reflected back to me, let's see if we can run JavaScript on it to steal a person's SessionID.

4. By Typing in <script>alert(document.cookie)</script> we get this output



Code:

Within the code inside purchase.php, lies the code that queries the database, when a user searches a term, we can see that if the $_GET variable has been set, we just use mysqli_query() to execute the statement which is very insecure because we are just passing the query string directly to the server without having any sort of sanitization, so the '$search$' term could be malicious javascript.

```
if($_SESSION["status"])
{
    $conn = connect_to_database();
    updateSessionCookie($conn, $_SESSION["username"]);

    $search = NULL;

    if(isset($_GET['search']))
    {
        $search = $_GET['search'];
        $query = "SELECT * FROM Movies WHERE Name LIKE '%$search%' ORDER BY Name";
        }
    else
    {
        // Get all the Movie Names
        $query = "SELECT * FROM Movies ORDER BY Name";
    }
    $result = mysqli_query($conn, $query)
 }
```

Next, is we can see by looking at the HTML inside the else statement is that User Input is being reflected back to the User which tells us the parameters which we send server side is coming back to the user's browser which will end up on the page sent to the browser so the browser will execute the malicious JavaScript.

```
<div class="col">
    <?php if($search == NULL)
    {
        echo '<h1 class="display-4" style="color:white">BUY MOVIES</h1>';
    }
    else
    {
        echo "<h1 class=\"display-4\" style=\"color:white\">Search: $search</h1>";
    }?>
```

Mitigation/Prevention Techniques:

Reflected XSS can be mitigated by just not having to reflect/show parameter values that the user sends back to the browsers thus your browser will never have to reload the JavaScript that was sent maliciously. Unfortunately, this is only a realistic for simple Web sites. In order to prevent XSS attacks from happening is to escape all the input by using php's "htmlspecialchars()" as a form of sanitization. Also by invoking HTTP-Only Cookies and the Same Origin Policy. So that the cookies cannot be accessed by JavaScript and only allow JavaScript to run from the same origin where the trusted site is and nowhere else.

2.XSS/Stored:

Description:

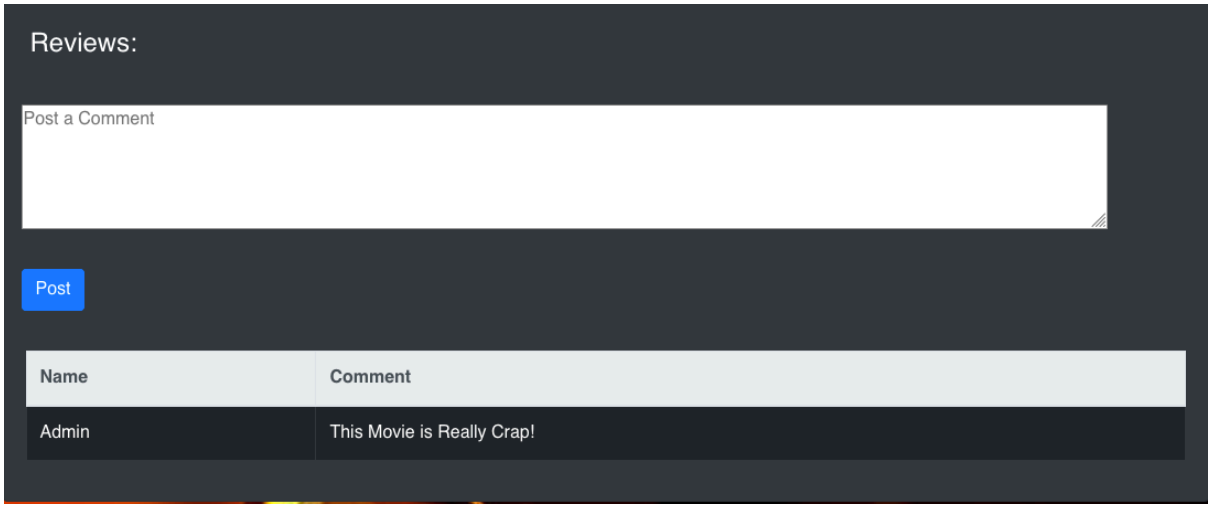
Cross-Site Scripting (XSS) is a different type of code injection vulnerability that exists over the web, this vulnerability is more severe than reflected XSS because it modifies the server and targets more users at once. Stored XSS occurs when a web application gathers input from a user which might be malicious, and then stores the input in the data store for later use. Stored XSS happens if this input is not filtered correctly.

Possible Exploits:

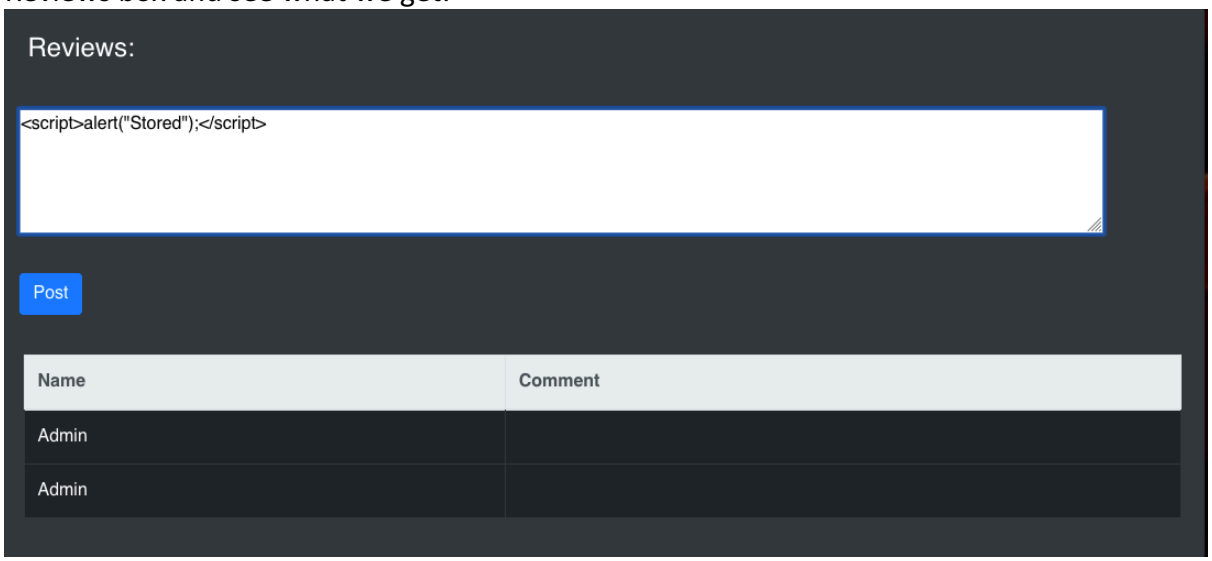
Similar to Reflected XSS this vulnerability can be used to do things such as session hijacking, capture sensitive information from anyone who accesses the webpage that has to load the malicious content and even port scanning dependent on the input.

Walkthrough:

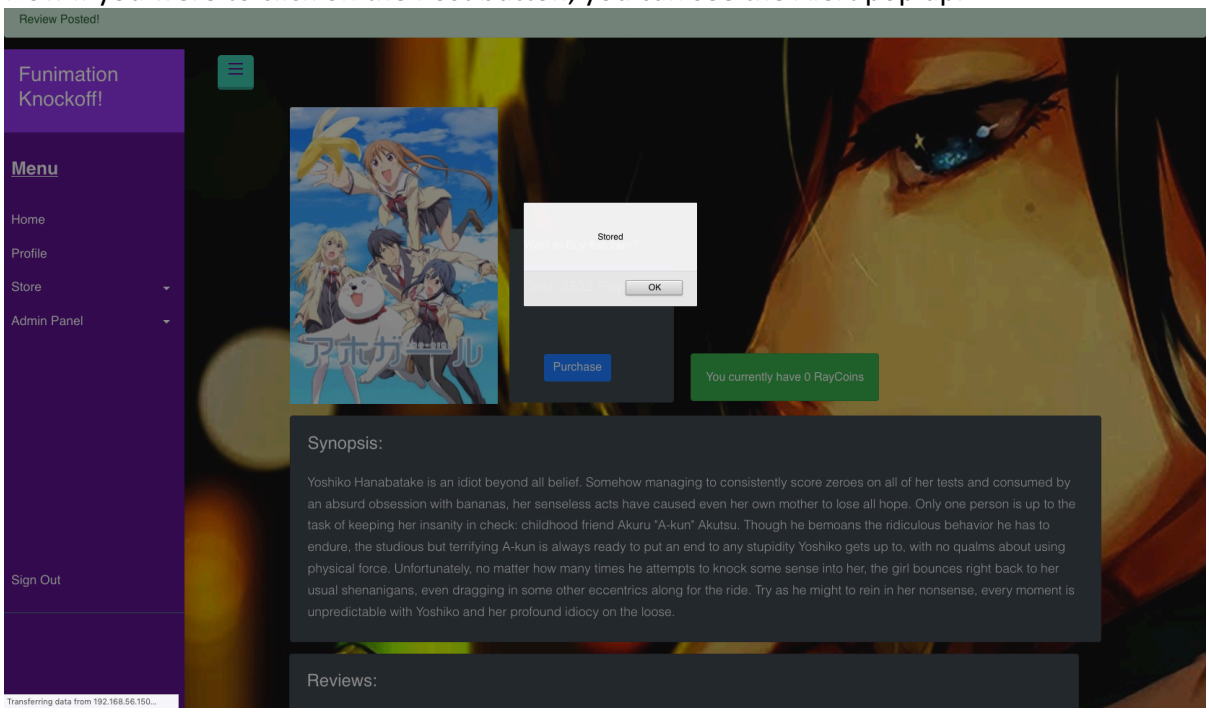
For this example, you would want to access any movie page from the purchase.php page. So, go to Purchase from the side panel and select on any movie which will redirect you to the movie page. Once at the movie page scroll down and find the Reviews table. Try and post a review yourself.



Once the review has been posted, let's try to inject some malicious JavaScript into the Reviews box and see what we get:



Now if you were to click on the Post button, you can see the Alert pop up.



Now try to refresh the webpage a few times, you can see the alert dialogue being shown again and again. This is because the malicious JavaScript has been stored into the Database and anybody who has access to this link, will be able to see the alert box pop up because it has been embedded into the database. This happens because the webpage loads all of the reviews upon access of the link including the malicious code.

Code:

```
// The User wants to post a comment on the movie
else if(isset($_POST['btn_comment']))
{
    // Can't Post Empty Reviews
    if($_POST["comment"] == "")
    {
        $_SESSION["error"] = "You must post something!";
        header("location: movie.php?select=$select");
        return;
    }
    else
    {
        // Query To Insert Comment into Reviews Table
        $query = "INSERT INTO Reviews(MovieID, UserID, Comment) VALUES($select, " .
$_SESSION["user_id"] . ", '" . $_POST["comment"] . "')";
        mysqli_query($conn, $query);

        // Notify User Comment was successfully posted
        $_SESSION["success"] = "Review Posted!";
        header("location: movie.php?select=$select");
        return;
    }
}
```

As you can see inside the else statement, there is no sanitization happening inside the query String and you are also joining Strings together. The only difference with the reflected XSS is that the User isn't reflecting the code back but the Database is executing the code as the page is being loaded every time, due to having to load the comments.

Mitigation/Prevention Techniques:

In order to mitigate stored XSS is the same as Reflected XSS, SANITIZE THE INPUT. Both at the Client Side and the Server Side.

3.SQL Injection

Description:

An SQL Injection attack consists of injecting SQL queries into the input data of an SQL query. SQL Injection allows the SQL query String to perform things that the was meant to be happening do something else. The idea of an SQL Injection is to try and dump the database back to the user/attacker so the attacker knows about certain sensitive information. Also with SQL Injection, you are able to modify/delete existing tables.

Possible Exploits:

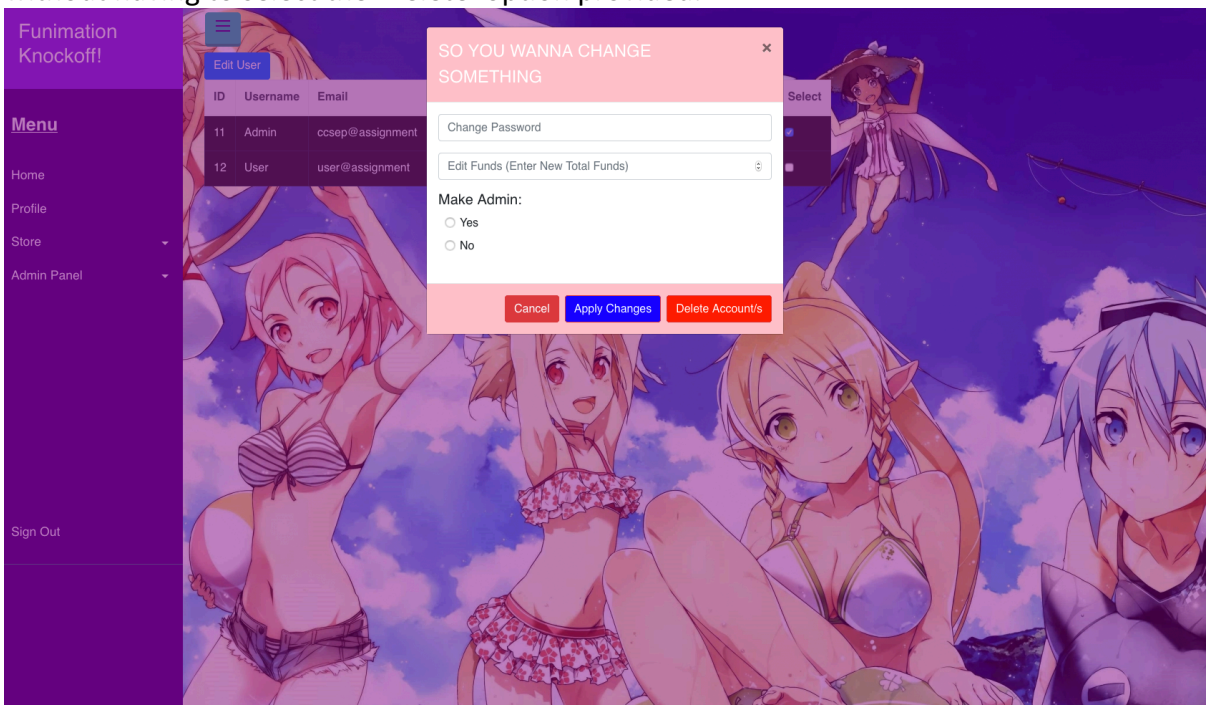
By using SQL Injection, you can exploit things such as modify/delete tables and steal sensitive information (e.g Identity theft) from the database. Even gain unauthorized access.

Walkthrough:

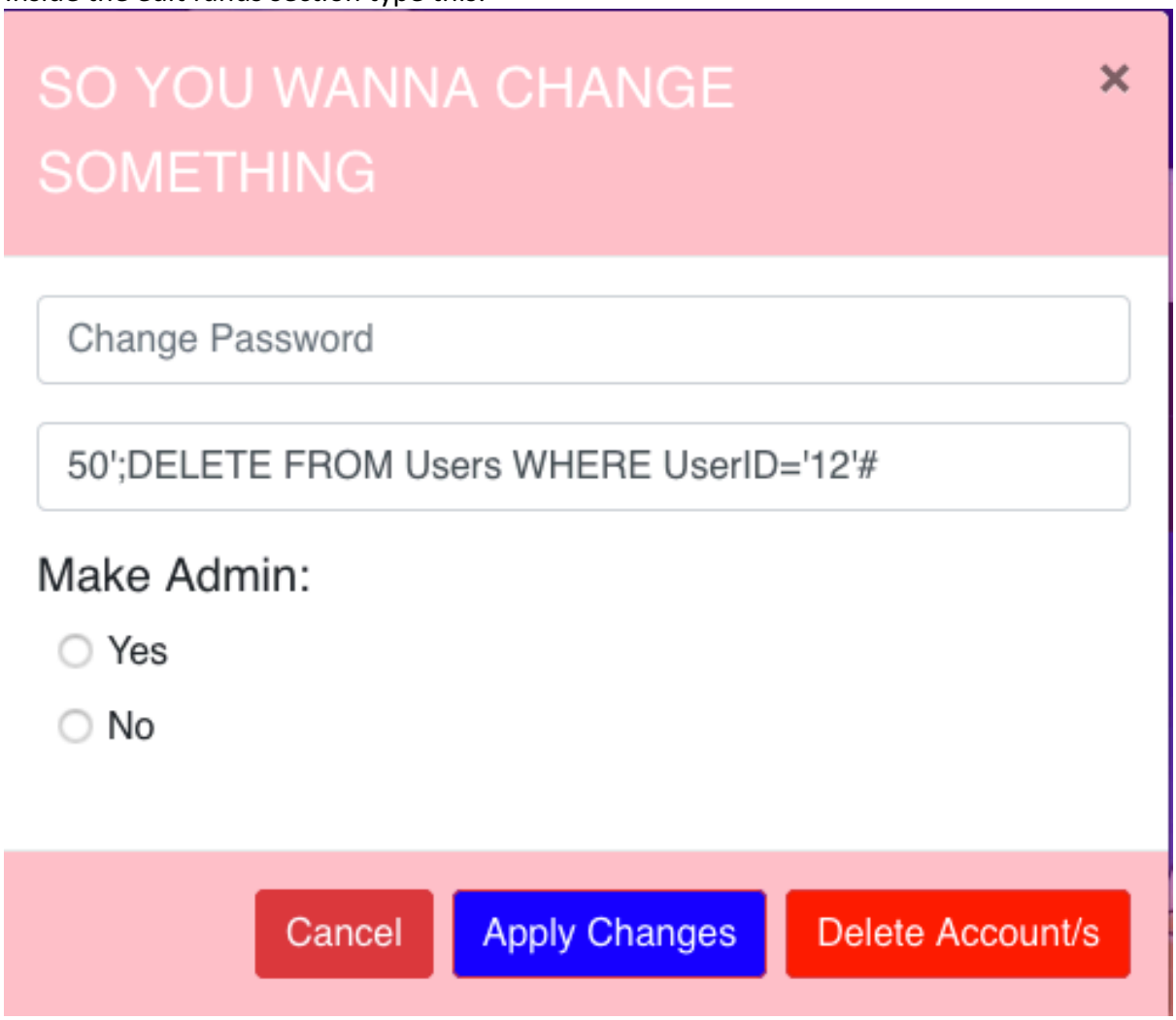
Throughout my code, there are a lot of attack vectors to do SQL Injection, (except the signup page because I used prepared statements there). But let us look at the “Manage User Accounts” tab which lives inside the Admin Panel to see what we can SQL Injection.

If you were to select any of the tick boxes and click on the Edit User button, you are able to edit specific User data. In this case, you can modify, the User’s password, change their funds or even change the privilege level. Try out the functionality to see if it works.

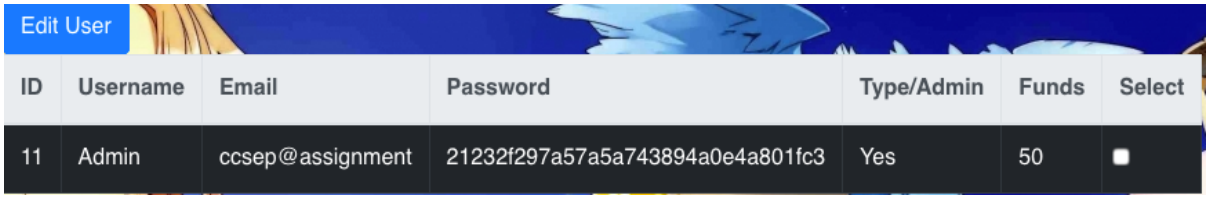
In this example, we are going to attempt to drop the User account from the Admin account, without having to select the ‘Delete’ option provided.



Inside the edit funds section type this:



So when you select the “Apply Changes” option it will set Admin’s funds as 20 and then DROP the User where the User ID is equal to 12. The # symbol is to comment any other legitimate SQL that is after. This SQL query takes advantage of the multi_query() function inside php where the code builds up a query string because you want to be able to change certain things but other things all together. This will be explained in more detail under the code section.



Code:

```
// Or the Apply Changes button was selected
else if(isset($_POST["apply_changes"]))
{
    // Begin building a Query String
    $query = "";

    if(!empty($_POST["change_password"]))
    {
        foreach($_POST['check_list'] as $check)
        {
            $query .= "UPDATE Users SET password='"
                      . md5($_POST["change_password"]) . "' WHERE UserID="
                      . $check . ";";
        }
    }
    if(!empty($_POST["change_funds"]))
    {
        foreach($_POST['check_list'] as $check)
        {
            $query .= "UPDATE Users SET funds='" . $_POST["change_funds"] .
                      "' WHERE UserID=" . $check . ";";
        }
    }
    if(!empty($_POST["make_admin"]))
    {
        if($_POST["make_admin"] == "yes")
        {
            foreach($_POST['check_list'] as $check)
            {
                $query .= "UPDATE Users SET type='0' WHERE UserID='"
                          . $check . "';";
            }
        }
        else if($_POST["make_admin"] == "no")
        {
            foreach($_POST['check_list'] as $check)
            {
                $query .= "UPDATE Users SET type='1' WHERE UserID='"
                          . $check . "';";

            }
        }
    }
    // Execute Query
    if($query != "")
    {
        // My reason to use multi query
        mysqli_multi_query($conn, $query);
    }
```

As you can tell from the code above, if the User were to select “Apply Changes”, there are a bunch of checks that happen to see if the User has typed anything inside the Modal boxes, if the user has, the query String is just appended and then executed at the very end of the query string using multi_query(). The problem with this is that there is no sanitization happening and we are joing Strings together with variables. Also, multi_query is a function that has been deprecated because it is a security flaw, that you should never have to have multiple queries executed at once if you want to have multiple queries executed, you should have several single queries that run alongside each other.

Mitigation/Prevention Techniques:

In order to mitigate SQL injections from happening, User input should be sanitized and you should never ever join Strings together. Also you should be running prepared Statements/Stored Procedures instead of php’s mysqli_query() and mysqli_multi_query() functions. By running prepared statements, you never trust user input with the original query together. Because prepared statements will treat User Input and the query String separately so SQL Injection is impossible.

4.SQL Injection/Blind

Description:

SQL Blind Injection is a type of SQL Injection, where instead of getting results directly from the database, you ask the database true/false questions and the web application will determine the response. With blind SQL Injection, you cannot do as much damage as normal SQL Injection but you can find programming security problems such as non-generic error messages that may give indicators to sensitive information.

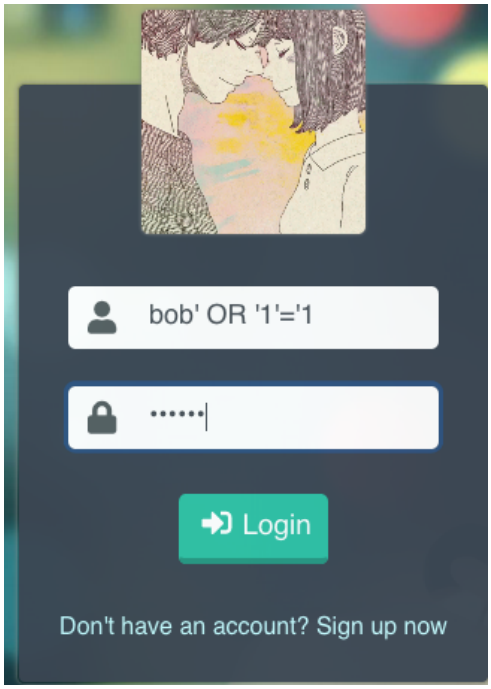
Possible Exploits:

Possible exploits for blind SQL Injection, is the same as SQL Injection except it is a lead up to those exploits.

Walkthrough:

For this example, we will use the login page as an example for Blind SQL Injection, so if you are already logged in as a user immediately log out and navigate to the login page.

1. Start off by typing some Random input into the Username and Password Sections,
    - Most people will find the error message will either say Incorrect Username or Incorrect Password (That is assuming you guessed the Username correctly)
2. This is very bad programming by the creator of the website, because if you output error messages of such, this implies that you hopefully got one of the input fields correct. Which might be enough information that a hacker can take of advantage of.
3. Now let’s type the following malicious SQL into the input fields.



Where bob can be any string and the 1=1 means it is always going to be true. The password field can be anything, if you get invalid password, you can tell that the program checks for Username first and then password.

Code:

```
/* This is where the Blind SQL Injection will be */
$query = "SELECT Username FROM Users WHERE Username='$user'";
$result = mysqli_query($conn, $query);
$count = mysqli_num_rows($result);

if($count == 0)     // Invalid User
{
    // Error Message using Flash Variable
    $_SESSION["error"] = "Incorrect Username";
    header("Location: login.php");
    return;
}
else if($count > 0)     // Valid Users
{
    // Now check if the given Username matches the password
   $query = "SELECT Password FROM Users WHERE Username='$user'
             AND Password='$pass'";
   $result = mysqli_query($conn, $query);
   $count = mysqli_num_rows($result);


   if($count != 1)
   {
```

```
        // Error Message using Flash Variable
        $_SESSION["error"] = "Incorrect Password";
        header("Location: login.php");
        return
    }
    else if($count == 1)    // Returns 1 row
    {
        // CREATE SESSION VARIABLES
        updateSessionCookie($conn, $user);

        // Allows the welcome message to be only shown upon login and never again
        $query = "SELECT username FROM Users WHERE username=?";
        $name = getRowValue($conn, $query, $user);
        $_SESSION["welcome_message"] = "Welcome {$name}!";

        header('Location: index.php');
        return;
    }
```

As you can see from the code above, you can tell that the error messages say Invalid Username or Invalid Password. Which gives the impression that it does the validation checking one after the other.

Mitigation/Prevention Techniques:

In order to prevent this, you can also do the same thing as SQL injection and use prepared statements for your queries but more importantly, to mitigate Blind SQL Injection you should have generic error messages, so you don't give any information back to the User. Error messages such as "Invalid Username/Password" will be fine if the User entered an incorrect Username or Incorrect Password, so the attacker won't know how the validation checking is done.

5.PHP File Include

Description:

PHP File Include takes advantage of the include() statement in php. Usually you use include to include local files into your code for modularity. In this case, the problem with Local File Inclusion is that since the file is localized. You are able to modify the request (Especially if it is inside the GET) to do a directory traversal to see what other files you have on the server. Also you could also put your own malicious file on too.

Possible Exploits:

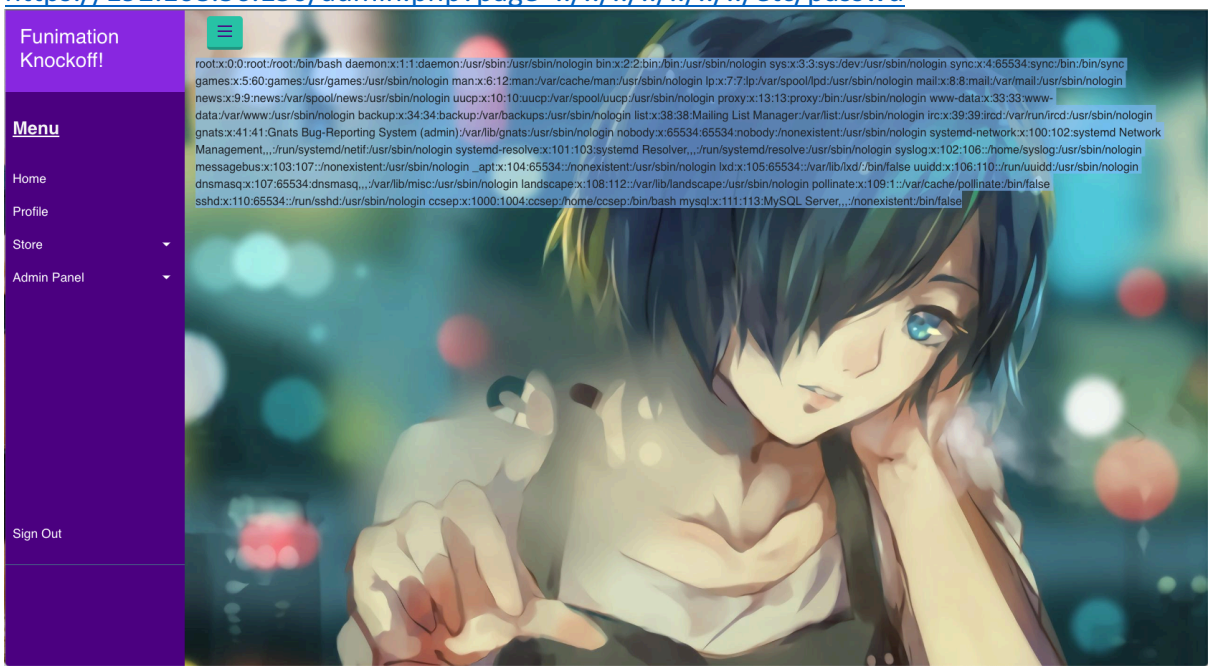
- Remote Code Execution
- Stealing of Personal Information

Walkthrough:

For this demonstration we are going to do two examples, both of these examples will be inside the admin panel. If we look into the “Add/Remove Movies” tab and look at the URL we can see the page uses Local File Include via a GET request. Let’s try to directory traversal all the way back to the root folder and see if we can get any sensitive information.

Before:
https://192.168.56.150/admin.php?page=movielisting.php

After:
https://192.168.56.150/admin.php?page=../../../../../../../etc/passwd



Oh Look! We just found the passwd file in where all the Usernames are stored.

For the second example we are going to use the “Manage User Accounts Page”. Inside the URL try type this:

https://192.168.56.150/admin.php?page=php://filter/convert.base64-encode/resource=database_con.php

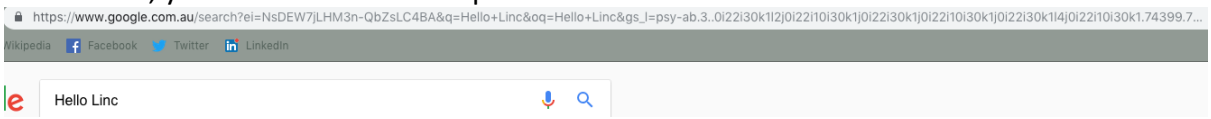
- This gives you the base 64 version of the php file database_con.php and if you were to throw the output into a base64 decoder, you will end up having the whole database_con.php source code.

Code:

The Code section for this is a bit too long to put into the report but if you have a look at the files in admin.php, userlisting.php and movielisting.php you can see at the top of admin.php the include of movielisting and userlisting is being included as a variable which is not good because this is where I will inject my code inside. admin.php exists simply to dynamically load the webpage so you won't have to report the same code for movielisting and userlisting.

Mitigation/Prevention Techniques:

Although, having your webpages load in dynamically can be really good from a Software Engineering standpoint (Re-use of code). Unfortunately, from a security perspective, it is not a very good idea. In order to protect from this, you could use character encoding to make it near impossible for an attacker to figure out what your directory is like. An example for this is if you were to type in the search bar in google.com, and look at the URL after you send the search., you will receive this output:



My search query was still executed, but everything after the '?' (GET REQUEST) is completely gibberish and an attacker will have a hard time trying to figure out your directory path if it has been encoded.

6.Broken Access Control

Description:

Broken Access control is how a web application grants access to content to certain users. The meaning of "Broken" is that via methods such as Insecure ID's, Path Traversal, File Permissions or Client Side Caching a normal user might have access to certain webpages that they aren't meant to see.

Walkthrough:

For this example, try and login as a Normal User and then in the URL bar, type in the code to access the admin panel i.e

- https://192.168.56.150/admin.php?page=userlisting.php
- https://192.168.56.150/admin.php?page=movielisting.php

If you can access the admin panel, that is Broken Access Control via File Traversal.
I have implanted some sort of security to this so that you are not allowed to file traversal if the User hasn't logged in. If you were to try to do this the webpage will redirect you back to the login page and *Kindly* ask you to log in.

Mitigation/Prevention Techniques:

There are several ways to mitigate this, the best way is to implement an Access Control Matrix, and then program your web application to follow that matrix. With an access control matrix you can define a set of access control rules where you won't what type of users are allowed to access which pages.

7.Use of hard-coded password:

Description:

Hard-coded passwords are a high severity risk because the programmer has physically implanted inside the code a password. The use of hard-coded password has many negative impacts, in which the most significant is the failure of authentication because if an attacker can get in with your password, they have the ability to do anything including changing the password. This also increases the risk of password guessing.

Walkthrough:

For hard-coded passwords, there is no walkthough that you can do this with, but you can find the hardcoded password if you look at the walkthrough for the php file include exploit and look inside the database_con.php after you have decoded the file. We are able to find the username and password that is used to go into the database inside that file.

Code:

```
function connect_to_database()
{
    $db_server = "localhost";
    $db_username = "ccsep";
    $db_password = "ccsep_2018";
    $db_name = "assignment";

    // Create Connection
    $conn = mysqli_connect($db_server, $db_username, $db_password, $db_name);

    // Check Connection
    if(!$conn)
    {
        die("Connection failed: " . mysqli_connect_error());
    }

    return $conn;
}
```

Mitigation/Prevention Techniques:

In order to prevent this is to not hard-code your passwords inside your code. You could create a script that inserts into the database so the password is nowhere present inside your code. Also asking the User for the password just to connect to the database helpe because at least the attacker if he gets that far into exploiting your program will have to guess the password.

8.Sensitive Data Exposure (Insecure Hash):

Description:

Sensitive Data Exposure, is a vulnerability when an application does not adequately protect sensitive information properly, this is a very broad topic because the data can range from anything such as insecure passwords, session tokens or database access. In this application the Sensitive data exposure happens in the encryption algorithm that generates the hash, which is MD5.

MD5 hashes are considered insecure because the hashing algorithm is considered too quick to encrypt, and if the encryption algorithm is considered too quick it is easy to break. Although hash algorithms are considered to be quick, if it is too quick you can craft malicious inputs and force hash collisions to happen. Because as long as the hash matches we can be authenticated.

Code:

The way all passwords are stored in my database are all by md5 hash, if you were to look inside my database_con.php file, signup.php inside my login.php file, you can see all of the passwords are using md5 hash algorithms stored into the database.

Mitigation/Prevention Techniques:

In order to prevent these weak hashes, you could use a different encryption algorithm that is more secure such as SHA-2 and SHA-3 hash functions.

9.Empty String Password:

Description:

Using empty string as a password is insecure, there is no case where it is appropriate simply because the password is too easy to guess. Especially nowadays where computers are fast enough to do brute force attacks

Walkthrough:

If you go to the signup page and signup as a new user, and you leave the password fields blank, you are able to register as a user with a blank password.

Code:

Inside the code, this happens inside the HTML reference, where you just comment the "required" field so you are able to submit a query with an empty string and it will work.

Mitigation/Prevention Techniques:

In order to mitigate this do not let the user enter an empty password and register that as valid. Have the required tags inside the html, but that isn't considered good enough because the client can alter that. In order to stop this, have the server check to see if the POST variable is set, and if it isn't ask the user to enter a password.

10.Missing Error Handling:

Description:

Missing Error Handling, is not defining or creating a custom error page when at least the 404 and 500 error codes arise. When an attacker tries to explore a web page, the amount of information that an exception exposes is crutial, if an attacker can find a stack trace from an error page you give the attacker a better understanding of what services run.

Walkthrough:

Go to the URL and type any random string after the '/' of the IP address. If you were to go to that address, you will see a 404 Not Found error. And from that message you can tell that the application is running off a Apache server on port 443.

Code:

There is no code to display

Mitigation/Prevention Techniques:

In order to prevent the leakage of information, you can create your own custom default error page, and inside the page just say an error has occurred but don't give away any information. In the php code you can just have an if() condition that checks the GET REQUEST and if the file doesn't exist just redirect to your own custom error page.

References:

https://www.owasp.org/index.php/Main_Page - For finding the vulnerabilities

setup.sql – Taken from Jonathon Winter in order to setup the database quickly
setupAssignment.sh - Taken and modifiedfrom Jonathon Winter in order to setup the database quickly
database.sql - Taken from Jonathon Winter in order to setup the database quickly