CCSEP Assignment

Jonathon Winter
Student ID: 1885 6204
Unit: Cyber Crime - Security Enhanced Programming

October 18, 2018

Contents

1 Overview

2 Vulnerabilities

2.1 XXS - Reflected

`https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)`

2.1.1 Description

Reflected XXS is a vulnerabilty where an attacker can inject code (typcially script tags) into a website via the input that they have entered. This is due to the fact that the website is updated accordinly to what a user has entered

The script is generally activated through a link that the attacker sends around in order to get clicks, and when clicked it will load the website and the malicous code

Lets say a website is seen as such http://www.injectable.com?name=John and when loaded it says "Hello John" then if I was to replace the value of name to <script>alert(document.cookie)</script> then when the page loads it will execute the script if it hasnt been cleaned first

2.1.2 Possible Exploits

The most common type of XXS Reflected attack is that once the link has loaded it will grab the users current session ID and send it to the attacker, with this the attacker can then log into the website and pretend to b that user.

In this situation however, in order to obtain a session ID the user must be logged in to the website already. In the case they arent then it will not work.

2.1.3 Location

The location of the vulnerability is in movies.php. As seen the user can enter a search string and it will no only display the matching movies however also display what the user had entered. So in this case when a script tag is entered it will add it to the HTML page and will be executed once it reaches the client.

```
<?php echo ($query != "All" && $query != '' ?  "$query" : "All" )?>
```

2.1.4 Instructions

1. Create malicous link.

   ```
   https://192.168.56.150/movies.php?query=a<script>alert(document.cookie)
   </script>
   ```

2. Send the link to someone to click.

2.1.5 Mitigate Strategy

The main method of fixing this would be to strip the tags and escape all escape characters from all input from the users that are reflected back to the user. The other method is to simply not reflect anything back to the website from user input, however this option isn't always available

2.2 XXS - Stored

`https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)`

2.2.1 Description

XXS Stored is very similar to XXS reflected however with the goal of being stored on the server, so when a usual client loads the page it will execute the script that an attacker has entered. This method, if can be done, if much more dangerous than XXS reflected since it ensured that the user is already using the website and no extra links have to be pressed in order to execute the malicouc script

2.2.2 Possible Exploits

The same type of exploit can occur for XSS reflected and stored. If the attacker is able to grab the users session ID and send it to the attacker then they can pretend to be them and log in as them. This is however only one of the infinite list of attacks that an attacker can do.

2.2.3 Location

This vulnerability is located in the reviews section of a movie. When a specific movie page is loaded, all of the reviews that have been submitted are also displayed at the bottom. If any tags are in someone's review, whether it be <b> bold tags, <i> italic tags or the infamous <script> tags they will be placed directly in. The file that includes this vulnerability is movie.php

Here is how your review is retrieved from the form:

```
$review = $_POST['review'];
```

And here is where the review is printed out (If it isn't your own review)

```
<?php echo $row['review']?>
```

2.2.4 Instructions

1. Either log in or register an account
2. Go to the movies page and purchase any movie
3. Once purchased the review block should appear for you to enter a review
4. Add your malicious script in this textarea and then submit review

   ex: This Movie was <script>alert(\'hi\')</script> really awesome
5. Log in as another user and go to the same move page. The script will then execute

2.2.5 Mitigate Strategy

This vulnerability can be mitigated by either allowing no tags in the review section and then using the *strip_tags*() function to remove them if present, or to only allow a select few tags that are designed just for styling, such as <b> and <i>

2.3 SQL Injection

`https://www.owasp.org/index.php/SQL_Injection`

2.3.1 Description

SQL Injection is an attack that consists of the attacker injection part of an SQL query into a query string to be executed. Depending on where the injection takes place, it could affect any type of query whether it be an insert, select, update or delete.

The overall goal of SQL Injection is to either gain access to information, ie. changing authorization level, or to cause damage to the system, ie. deleting values.

2.3.2 Possible Exploits

By being able to execute an SQL injection the attacker can modify fields that that normally are not able to. Depending on how the queries are set up they could easily do ";DROP Tables#" and then whoops, everything has been deleted. Not all attacks are going to be this detrimental, however attackers do find other ways to gain information that they shouldn't have, or to cause harm to the system.

If the attacker finds a field that they can use to update their own user details, then they could possibly use it to update more details that they normally don't have access to.

Example shown below in Instructions

2.3.3 Location

The type of SQL Injection that is mainly accessibly by all is an UPDATE query that modifies details about the current user. The vulnerability is located at `https://192.168.56.150/account.php?view=details`.

The value inside the newBalance field is used to directly modify the balance of the current user by executing the SQL query UPDATE. The query template is:

```
$sql = "UPDATE Users SET balance='".$_POST['newBalance']."'
          WHERE userID = '$userID'";
```

Since the value of newBalance cannot directly be edited by the user it is believed to be safe and is used directly in the middle of the query. IF the value of newBalance was to somehow be maliciously set then the query could be corrupted.

2.3.4 Instructions

1. go to `https://192.168.56.150/account.php?view=details`
2. open up developer tools (F12) and locate the input field for "new Balance"
3. remove the readonly tag and change type="number" to type="string"

This allows you to modify the value inside the input field directly as before it was set to be readonly and could only contains numbers, if anything else was inserted then it wouldn't submit the form.

4. Change the value of "New Balance" to the malicious SQL query
    - 500', access='admin

Sets by balance to 500 and my access level to admin (reload the page or goto another page to see the admin panel now appear)

- 0', where '1'='1'#

Sets the balance of all users to 0

2.3.5 Mitigate Strategy

There are 2 main methods for fixing this type of vulnerability

1. Prepared statements
2. Escape special characters (',",#...)

Prepared statements: A prepared statement is a very similar to an SQL query however a template is first build and prepared via the database and then is called upon when requested by filling in the missing values. This method is a simple solution as it not only allows the use of the same query multiple times but removes the need to escape any values that will be used.

Escape special characters: Values such as ', ", # and so forth can be used to manipulate an sql query and make it append new sections or completely cut it off early via the comment character. By first checking the value that is to be placed into the query, either removing the escape characters or by putting a \in front of them to make them redundant. This method is not foolproof though as there can be some methods around it and therefore if possible, prepared statements are probably the best fit.

2.4 SQL Injection Blind

`https://www.owasp.org/index.php/SQL_Injection`

2.4.1 Description

A blind SQL injection is very similar to the normal SQL injection attack where the goal is to inject data into an sql query to get more information or manipulate the system, however blind SQL has a different method of doing this.

Blind SQL is a method used to query a database when the only results that you are able to see are true or false. So if you were to query 1=1 it would return true or an appropriate message, however change it to ask if 1=2 then it would return false. While it may seen tedious, after a certain amount of querying and asking filtering, information on the whole database can be figured out.

2.4.2 Possible Exploits

While a simple true or false return value doesn't seem to bad, imagine if they started asking if the name of the database matches 'x' or if it contains 'y'. After a few attempts of this they will get to know the database name. Ok cool what's next, lets see if we can enumerate character by character what the table names are, next we can go for the columns in each table, and well would you look at that, they now have the whole layout of an entire database with all its tables and columns sorted out.

2.4.3 Location

The location of this vulnerability in the website is at the login page. `https://192.168.56.150/login.php`, but to be more precise it is in the file server.php. If a incorrect username is entered then the error states so, however if a valid username is used with an incorrect password then the error states a different prompt.

By realising this, an attacker than then check if the username field is injectable and vulnerable to SQL injection

```
$sql = "SELECT * FROM Users WHERE username = '$username'";
$results = mysqli_query($db,$sql);

/*...*/

$count = mysqli_num_rows($results);

if($count > 0){
    $row = mysqli_fetch_assoc($results);
    if(password_verify($password, $row['password'])){
        /*Redirect to login*/
    }else
        array_push($errors, "Invalid password for user: $username");
}else
    array_push($errors, "No user exists with that username");
```

2.4.4 Instructions

Enter these values into the username field and press submit. The value of the password field is irrelevant

1. hello' OR '1'='1'#
2. hello' OR '1'='2'#

Very simple examples to show that the field allows multiple checks in it. The ' are used to escape the current string. Which quotes to use can be determined though multiple attempts to see which ones work

3. hello' OR database() = 'assignment'#
4. hello' OR database() = 'HelloWorld'#

Returns true and then false, this shows that the name of the database is assignment. Obviously guessing this in one hit is quite unlikely and the multiple questions would be asked before this is discovered.

2.4.5 Mitigate Strategy

Just like normal SQL injections, the same methods of mitigating it can be used, this therefore includes prepared statements and escape escape characters. The other method that can be used to prevent Blind SQL injection is to generalize the errors. In this example change the error message for invalid username and invalid password to be the same message "Incorrect username or password" disregarding what actaully caused the unsuccessful login.

2.5 PHP File Include

`https://www.owasp.org/index.php/PHP_File_Inclusion`

2.5.1 Description

In PHP and many other languages the functionality to include a file to be used in the current file is used to simplify and reuse snippets of code or design. The use of it is used in many situations for valid reasons, such as can be seen in my implementation where I have set it up where every page first loads a verifyLogin.php file which checks that a session is currently active and if not then redirects to the login.

2.5.2 Possible Exploits

In the case stated above the file that is to be loaded is purely hard-coded and therefore can most likely not be changed by the user, however what is the file that was included was dependant on what a user has entered or what a GET or POST variable was. If the attacker was to manipulate the value of that variable then a they could potentially load any file on the system, once they figured out what the include command looks like.

2.5.3 Location

The location of this exploit in my website is in the account.php page. When loaded a noticeable variable is appended to the URL string and by default is "view=details". This by itself seems fine, however when the page is changed to the purchased movies or the reviews section then the value of view is changed to purchased and reviews respectively while the url is still https://192.168.56.150/account.php. This shows that the content shown is most likely determined by a include function which uses the view variable. If the value of view is manually changed to something invalid then the body contents it empty.

```
/*If view page is set then use that one*/
if(isset($_GET['view'])){
    $view = $_GET['view'];
}else{/*Else just use the default one*/
    $view = "details";
}

/* ... */

<div class="container">

    <?php include("$view.php"); ?>

</div>
```

2.5.4 Instructions

1. To show that it works: `https://192.168.56.150/account.php?view=`
2. To include another page:`https://192.168.56.150/account.php?view=includeme`
3. To extract another php page:

- Goto: `https://192.168.56.150/account.php?view=php://filter/convert.base64-encode/resource=account`
- Copy the base64 text that appears
- Decode it and viola you have a copy of the php file

2.5.5 Mitigate Strategy

To help in mitigating the potential damage from a file include a few settings can be changed to assist in it, the first one being allow_url_include. By turning this off it removes the possibility for any file include to be linked to another page that isn't on the host machine, meaning you can no longer run an evil php page or so forth. The other setting to turn off would be allow_url_fopen which once turned off just adds another layer of protecting from slightly different file includes from a remote server.

In the situation such as my implementation where it can be used to include a local file, a very simple method of restriction has already been used. The .php extension is added manually in when the file is to be included, and because of this files such as passwd cannot be extracted. However this doesnt stop them from including other php files and if a malicious php file was to end up in the system then they can do whatever they wish.

To add another slight level of protection against the base64 filter, by appending a directory path infront of the file to be include.

```
<?php include("./$view.php"); ?>
```

In this case aswell, their are only 3 viable pages that should ever be the value of view, therefore when first extracting the value of $_GET['view'], verify that it is 1 of those 3, else disregard it

2.6 Broken Access Controls

`https://www.owasp.org/index.php/Broken_Access_Control`

2.6.1 Description

Access control is used to determine if the level of authorization of the current user has correct permissions to gain access to something, however if these settings are not set up correctly then unauthorised accesses and modifications can occur.

2.6.2 Possible Exploits

If access control is not set up correctly then access to pages and information may be done while either unauthorised or not even logged-in. In a situation like this, an attacker may gain information that they would not normally have access to and in many cases be able to modify values that are only available to users or higher privileges, usually an admin.

2.6.3 Location

The location of this exploit is quite obvious however is a implementation that occurs to often. When a user is logged in and has either access level or admin or moderator, they are able to view the admin section in the navbar and use that to navigate to the relevant pages to alter both movies and users. In the case when a normal user is logged in the admin section is no longer visible and no sign of it is available to the user, so it should be safe and hidden correct?

If you navigate to the url `http://192.168.56.150/admin` then you shall see that there is a folder called admin and inside of it is 2 very specific pages, editUsers and editMovies. If the pages are tried to be accessed it will return the user back to the login page to start a session, however in the case a user is currently logged in then it will give them direct access to the page irrelevant of what their actual permission are.

While there is they go into editUsers.php then they will be able to not only modify their details but the details of any other users, this includes admin.

2.6.4 Instructions

1. Log in or register an account
2. Goto `https://192.168.56.150/admin`
3. Select one of the two php pages that have appeared
4. Viola you now have access to information and modification power that only admin and moderators should have. Enjoy.

2.6.5 Mitigate Strategy

Obviously the pages cannot be removed, so therefore further protection must be added to them. A simple method would be to verify the access level of the user when the page is first requested and if the access level doesnt match the required level then to display a 404 Page not found error. By displaying a 404 error it makes it appear as it the folder and files don't exist and aren't just restricted

Another level of protection to add would be that at the time any modifications are made double check the users access level directly from the database and not from the session. That way even if they did get to the page, they cant do anything. This method only reduces the damage, as the attacker would still be able to view the page and its contents.

3 Other Vulnerabilities

3.1 Unrestricted File Upload

`https://www.owasp.org/index.php/Unrestricted_File_Upload`

3.1.1 Description

Uploading a file can be used for many useful purposes, such as data storage, profile images and for sharing with others. However if the contents of what's in the file isn't checked or if no restrictions are added then in many cases you are opening up your system for the world to take

3.1.2 Possible Exploits

If a system allows any and all files to be uploaded to the server then in most cases, such as a website, they are open to many types of attacks. If a php file was uploaded to the server, and stored in the directory of the webpage for example, then I could add any code I wanted to that page and it will be executed once I open that new webpage.

This allows an attacker to have an endless supply of potential exploits that they can run, it could be from running a custom sql query to displaying the contents of the folder hierarchy or to take it to the next level, give themselves a reverse shell straight into the system.

3.1.3 Location

The main location of this exploit is in the details page of the account section: `https://192.168.56.150/account.php`. Once logged in and navigated to that page a user can upload a profile image to be used.

The server side verification of the file is only the minimum amount needed to verify that the file is valid. No checks for file types or valid extensions are done only that the file exists and that it is less than 2MB of storage (Default PHP maximum upload size). Once verified the users profile image is set to the path of the new file and while the name is changed to the usersID to prevent overwriting another's users upload, the extension is still maintained

details.php:

```
if(file_exists($_FILES['newPhoto']['tmp_name'][0])){
    $file_tmp =$_FILES['newPhoto']['tmp_name'];
    $file_type=$_FILES['newPhoto']['type'];
    $file_ext_tmp=explode('.',$_FILES['newPhoto']['name']);
    $file_ext=strtolower(end($file_ext_tmp));

    /*Change the filename to the UserID but maintain the extension*/
    /*A list of allowed extensions need to be created. ATM just allow
        them all*/
    $file_name = $userID.".".$file_ext;

    /*Delete the current profile pic*/
    shell_exec("rm img/profile/$userID.* 2>/dev/null");

    /*Upload the file*/
    move_uploaded_file($file_tmp,"img/profile/".$file_name);

    $sql = "UPDATE Users SET image = '$file_name' WHERE userID = '
        $userID'";

    mysqli_query($db, $sql);

    /*Set the new image for the image locally for the time being*/
    $user['image']=$file_name;
}
```

3.1.4 Instructions

The following guide is just a proof of concept that this works. no malicious content is in the file to be uploaded, however can easily be changed to cause a reverse shell (has been tested)

1. Obtain or create a simple php file. Only needs to have a single line in it such as <h1>I did it</h1>
2. Log in or register an account: `https://192.168.56.150/login.php`
3. Goto `https://192.168.56.150/account.php`
4. Select to upload an the custom php file

5. After uploading navigate to `https://192.168.56.150/img/profile/x.php` where 'x' is your userID.
6. Your custom php should now be previewed.

3.1.5 Mitigate Strategy

There are 2 main methods that can be used to fix this issue.

1. Restrict file uploads to be of a certain or range or extensions

When the file is uploaded check if the extension of the upload is an allowed extension. In order to do this a whitelist of image types but be sorted out. For example just allow jpg and png files and if the file is anything else then reject it, even it is possibly an image.

2. Remove the complete feature of profile pictures (not always feasible)

If the use of a file upload isn't required and causes no beneficial advantage to the system then the easiest way to reduce the chance of damage is to completely remove the functionality. Obviously this can not always be done and the previous method should be used.

3.2 Empty Password

`https://www.owasp.org/index.php/Empty_String_Password`

3.2.1 Description

Allowing the use of an empty password for a user when registering and logging in to the system.

3.2.2 Possible Exploits

By allowing an empty password to be used for an account, it removes any attempt to ensure that users are using strong passwords and allow an attacker to get straight access to a user by only knowing their usernames.

Since a username is displayed in both the users.php page for admin and if they have made a review on the movie then getting a list of usernames isn't that hard.

3.2.3 Location

Empty passwords can be set when registering an account and used when logging in. They can also be set at the update password section in the account page. The only validation done is that they are the same

```
...

$password1 = cleanInput($_POST['password1']);
$password2 = cleanInput($_POST['password2']);

...

if($password1 != $password2)
    array_push($registerErrors, "Your passwords do not match");

...
```

3.2.4 Instructions

- Go to `https://192.168.56.150/login.php`
- Use username: Chris Chang
- Leave password field empty
- Press login.
- Congrats, you're in.

3.2.5 Mitigate Strategy

Set in place a password policy where it must be x characters long and have x amount of special characters. That way it reduces the change of a users password being brute forced and a user having a extremely insecure password. 2 quick modifications can be done though to the current code

1. Make is so that a password has to be entered. Add the "required" value

```
<input type="password" name="password1" id="password1" required>
```

2. Ensure client side that the password isn't empty (Can never trust the client)

```
if(empty($password1))
    array_push($registerErrors, "Password cannot be empty");
```

3.3 Using a broken or risky cryptographic algorithm

`https://www.owasp.org/index.php/Using_a_broken_or_risky_cryptographic_algorithm`

3.3.1 Description

When trying to protect data or obscure it to hide the original value, a cryptographic algorithm is usually used, however if there is an issue with the algorithm used then it can potentially be reversed enginerred and what was once believed to be secret is now known by most likely someone who shouldnt see it. If the algorithm used is infact broken/exploitable then using it will have the same effect as if the passwords were written in plain text in the source code of the website. While not everyone will notice it there, once they inspect it they will be able to get the password without much struggle.

3.3.2 Possible Exploits

In the example of this website, the value that has a cryptographic algorithm to secure it is the password of a user. Not if an attacker was to obtain the hashed1 password, then they will be able to log in as that user and pretend to be that user while using the system, therefore hiding the fact that it was actually them logged in. In some cases if people use the same password for multiple accounts, then it also allows them to try gain access to other websites and applications that they use.

In the case where a users password is obtained then you have broken both your confidentiality of the website and also your integrity, due to allowing someone other than the actual owner of the account to act as if they were them.

3.3.3 Location

The location of this exploit is obviously the password field for the users. The cryptographic algorithm currently used is md5 of which was invented back in 1991, however was proved broken in 2012. Therefore using it now is quite innefective due to the range of tools that are now available to crack these hashes

```
/* Registering */

$hashPassord = md5($password1);

$query = "INSERT INTO Users (username, password, email, name)
    VALUES ('$username', '$hashPassord', '$email', '$name')";

...

/* Logging in */

if(md5($password) == $row['password']){
    /* Proceed to home page */
}
else{
    /* Display incorrect password */
}
```

3.3.4 Instructions

- Login as admin:admin
- Goto `https://192.168.56.150/admin/viewTables.php?table=Users`
- Copy any of the hashed passwords
- Decrypt it as `www.md5online.org`

3.3.5 Mitigate Strategy

The most simplest solution is just to change the cryptographic algorithm that is being used, and luckily PHP has your back in this case and has an option to hash a string with the the "Default" hash. Meaning it a currently used hashing algorithm that is still known to be safe. If the code is changed to:

```
$hashPassword = password_hash($password, PASSWORD_DEFAULT);
```

And then when it comes to verifying it you instead do

```
if(password_verify($password, $hashPassword)){
    /* Passwords match*/
}else{
    /* Passwords dont match */
}
```

3.4 Multiple Admin Levels

`https://www.owasp.org/index.php/Multiple_admin_levels`

3.4.1 Description

In an application or website where administration privileges are implemented, in some cases it is required to have multiple levels of admin. For example you would have 3 possible levels

1. Admin: Can do anything they want
2. Moderator: Has restricted admin privileges but can still access some restricted content. In this case they cant add or remove, only edit.
3. User: The normal access level for a user.

3.4.2 Possible Exploits

The vulnerability comes into place when a user with certain access level can use their privileges to modify themselves to a higher level or modify a user that already has a higher access level.

3.4.3 Location

When viewing the edit Users page the list of users and there admin privileges can be seen, they can also be changed. By being a moderator you are not allowed to remove or delete users and movies, however that doesnt prevent you from changing the permissions of either yourself or another user (ie admin).

3.4.4 Instructions

- Log in as moderator:moderator
- Goto `https://192.168.56.150/admin/editUsers.php`
- Modify moderator and set Access to admin
- Congrats you just escalated your own privileges

3.4.5 Mitigate Strategy

There many different strategies to assist in mitigating this vulnerability however they are all dependant on what the system requires, One such solution in the example of this website would be to, if the access level is moderator, then only show users whos access level is lower then your own (in this case access:user).

However in the case where a moderator still needs to be able to see all users irrelevant of their access level, such as the current implementation, then a defence needs to be added to block moderators and even user access level from modifying those of equal or higher level then them. Solution would be to store a ranking system that determines if you have high enough access level and if required level isn't reached then disable modification features for that user, however still allow them to view that that user exists in the system.

```
/*UserID user being modified*/
$modUserID = $_POST["userID"];

$sql = "SELECT access FROM Users WHERE userID = $modUserID";

/*Current access of user being modified*/
$modCurrentAccess = mysqli_fetch_assoc(mysqli_query($db, $sql))['
    access'];

/*Access of user logged in*/
$modAccess = $_POST["access"];

/*If your access is user you cant alter anyone*/
if($_SESSION['access'] == 'user')
   return
/*If access is moderator you can only modify accounts to and from
    users and moderator access*/
if($_SESSION['access'] == 'moderator' && ($modCurrentAccess == 'admin'
     || $modAccess == 'admin')
   return
```

3.5 Password Management: Hard-coded Password

https://www.owasp.org/index.php/Password_Management:_Hardcoded_Password

3.5.1 Description

Hard-coded passwords are a vulnerability that are pretty self explanatory, somewhere in the code or configuration a password is written in plain text and if someone was to get access to the location of it they can read the password as it is.

3.5.2 Possible Exploits

If you can get access or view the file that has the password hard-coded for the MySQL connection, then you are able to retrieve it and use it as you wish. You can then attempt to connect and query the database manually without going through the website.

3.5.3 Location

In this implementation the password that has been hard-coded is the database password. If you open up the file php/config.php then you will see it there. If you try to get to the link manually you'll notice it is empty so it must be safe right? Nope.

```
define('DB_SERVER', 'localhost');
define('DB_USERNAME', 'ccsep');
define('DB_PASSWORD', 'ccsep_2018');
define('DB_DATABASE', 'assignment');
$db = mysqli_connect(DB_SERVER,DB_USERNAME,DB_PASSWORD,DB_DATABASE);
```

3.5.4 Instructions

These instructions will show how to get the config.php file from the web server and view the password from the client side. This method works due to also having the File Include vulnerability as stated before

- Goto `https://192.168.56.150/account.php?view=php://filter/convert.base64-encode/resource=php/config`
- Copy the base64 text and decode it
- View the contents that you have received
- You now have all the connection information

3.5.5 Mitigate Strategy

The main issue with this vulnerability is that majority of the time, the connection needs to somehow know the password to use for the connection. So instead of removing it you mostly just need to try protect it.

A simple yet not very effective solution is to move the configuration file outside the wwwroot directory. That way they cannot get direct access to it, and if the systems permissions are setup correctly then they cant get indirect access due to user permissions.

The instructions used to get the configuration file only worked due to the fact that the file is appended with a php extension. Therefore another method of mitigating this vulnerability is to place the sql setting in a .ini file and then load it. And then to prevent direct access to this file use a .htaccess to set the permissions for that file to Deny from All

4 Instructions

No modifications have been don't to the VM. In order to install the vm go to the root directory of the submission and run the install script. ./setupAssignmnet

Warning: This will delete any files currently in the /var/www/html/ directory and drop the current assignment database.

5 Known Defects

All the listed vulnerabilities were ones that I intentionally implemented and located in the website, there may be and is most likely other locations where vulnerabilities may exist. For example no sql injection or xss vulnerabilities were tested in the admin pages, however they are most likely present there is tested.

The website itself however works as intended and all requested features can be used, and if a regular user what to try and use the website, the chance of them occidentally running into a vulnerability is limited

A vulnerability that could've been discussed is the fact that no logging was done on the website, however this was ignored due to the fact that it was already done by me not having to do anything to create it.

References

[mal] myanimelist. *Anime and Manga Database and Community*. URL: https://myanimelist.net/ (visited on 10/2018).

[bs4] Mark Otto and Jacob Thornton. *Introduction to Bootstrap 4*. URL: https://getbootstrap.com/docs/4.1/getting-started/introduction/ (visited on 10/2018).

[owaa] owasp. *Empty String Password*. URL: https://www.owasp.org/index.php/Multiple_admin_levels (visited on 10/2018).

[owab] owasp. *Empty String Password*. URL: https://www.owasp.org/index.php/Password_Management:_Hardcoded_Password (visited on 10/2018).

[owac] owasp. *Empty String Password*. URL: https://www.owasp.org/index.php/Empty_String_Password (visited on 10/2018).

[owad] owasp. *Empty String Password*. URL: https://www.owasp.org/index.php/Using_a_broken_or_risky_cryptographic_algorithm (visited on 10/2018).

[owae] owasp. *Empty String Password*. URL: https://www.owasp.org/index.php/Unrestricted_File_Upload (visited on 10/2018).

[owaf] owasp. *Empty String Password*. URL: https://www.owasp.org/index.php/PHP_File_Inclusion (visited on 10/2018).

[owag] owasp. *Empty String Password*. URL: https://www.owasp.org/index.php/Cross-site_Scripting_(XSS) (visited on 10/2018).

[owah] owasp. *Empty String Password*. URL: https://www.owasp.org/index.php/SQL_Injection (visited on 10/2018).

[mpf] phpknowhow. *MySQLi Procedural Functions*. URL: https://www.phpknowhow.com/mysql/mysqli-procedural-functions/ (visited on 10/2018).