Assignment II – Preventing Web Application Vulnerabilities

CPS 499-02/592-02

Software/Language Based Security

Fall 2020

Dr. Phu Phung

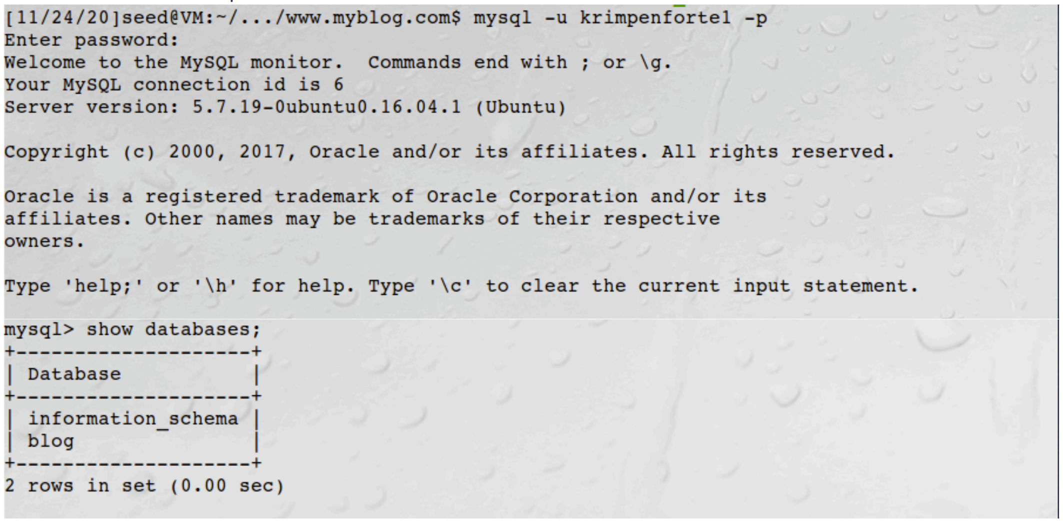
Evan Krimpenfort

URL: https://github.com/Krimpenfort23/autumn-2020/tree/master/assignments/assignment2

Task 0: Web Administration – Preparation

a. Database Setup

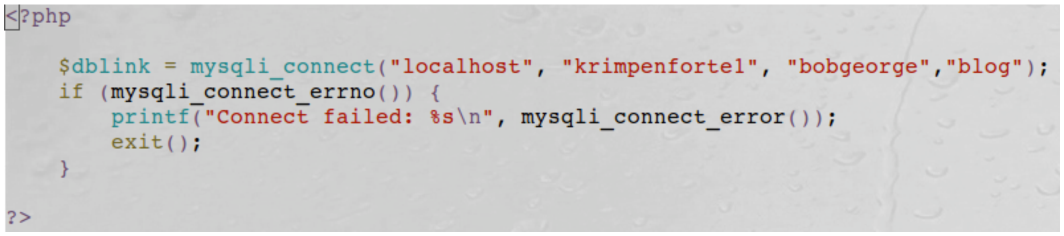
i. Imported the database and created a new user



```
[11/24/20]seed@VM:~/.../www.myblog.com$ mysql -u krimpenfortel -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.7.19-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| blog               |
+--------------------+
2 rows in set (0.00 sec)
```

Figure 1: The database

ii. Demonstration

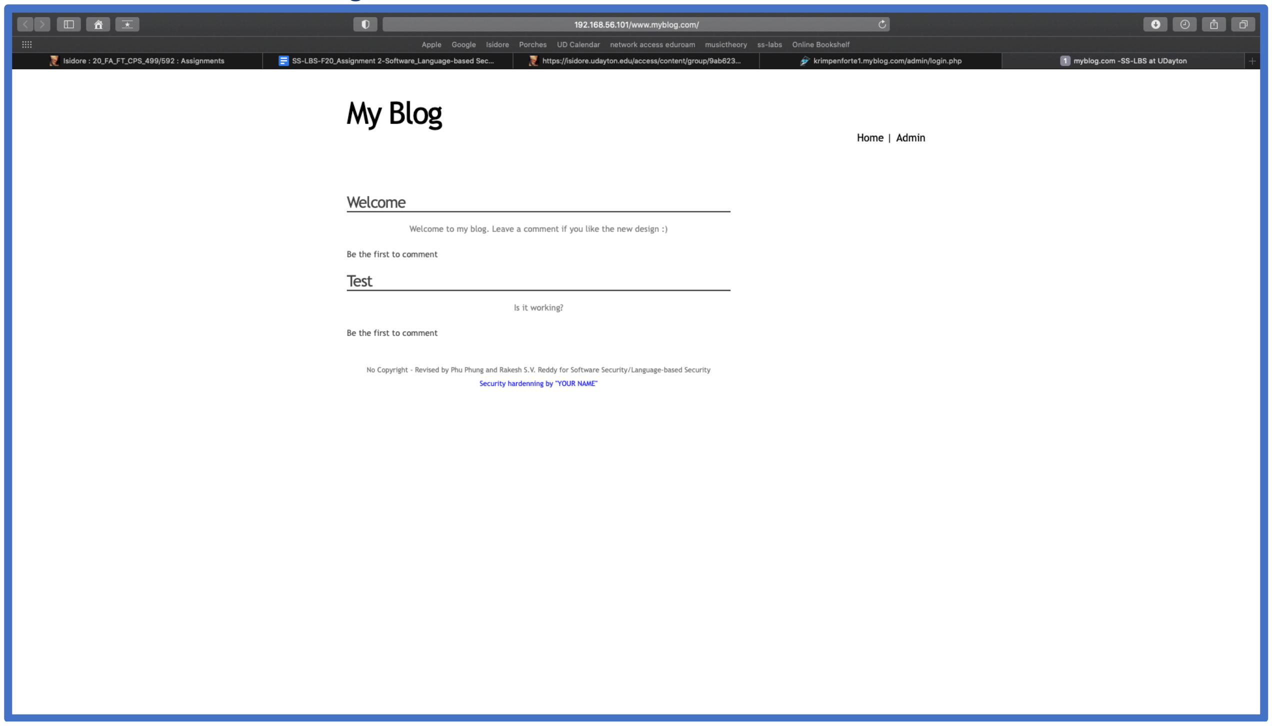


```
<?php

    $dblink = mysqli_connect("localhost", "krimpenfortel", "bobgeorge","blog");
    if (mysqli_connect_errno()) {
        printf("Connect failed: %s\n", mysqli_connect_error());
        exit();
    }

?>
```

Figure 2: classes/db.php

b. Deployment

iii. Using IP Address


192.168.56.101/www.myblog.com/
Apple Google Isidore Porches UD Calendar network access eduroam musictheory ss-labs Online Bookshelf
Isidore : 20_FA_FT_CPS_499/592 : Assignments
SS-LBS-F20_Assignment 2-Software_Language-based Sec...
https://isidore.udayton.edu/access/content/group/9ab623...
krimpenforte1.myblog.com/admin/login.php
myblog.com -SS-LBS at UDayton
Home | Admin

My Blog

Welcome

Welcome to my blog. Leave a comment if you like the new design :)

Be the first to comment

Test

Is it working?

Be the first to comment

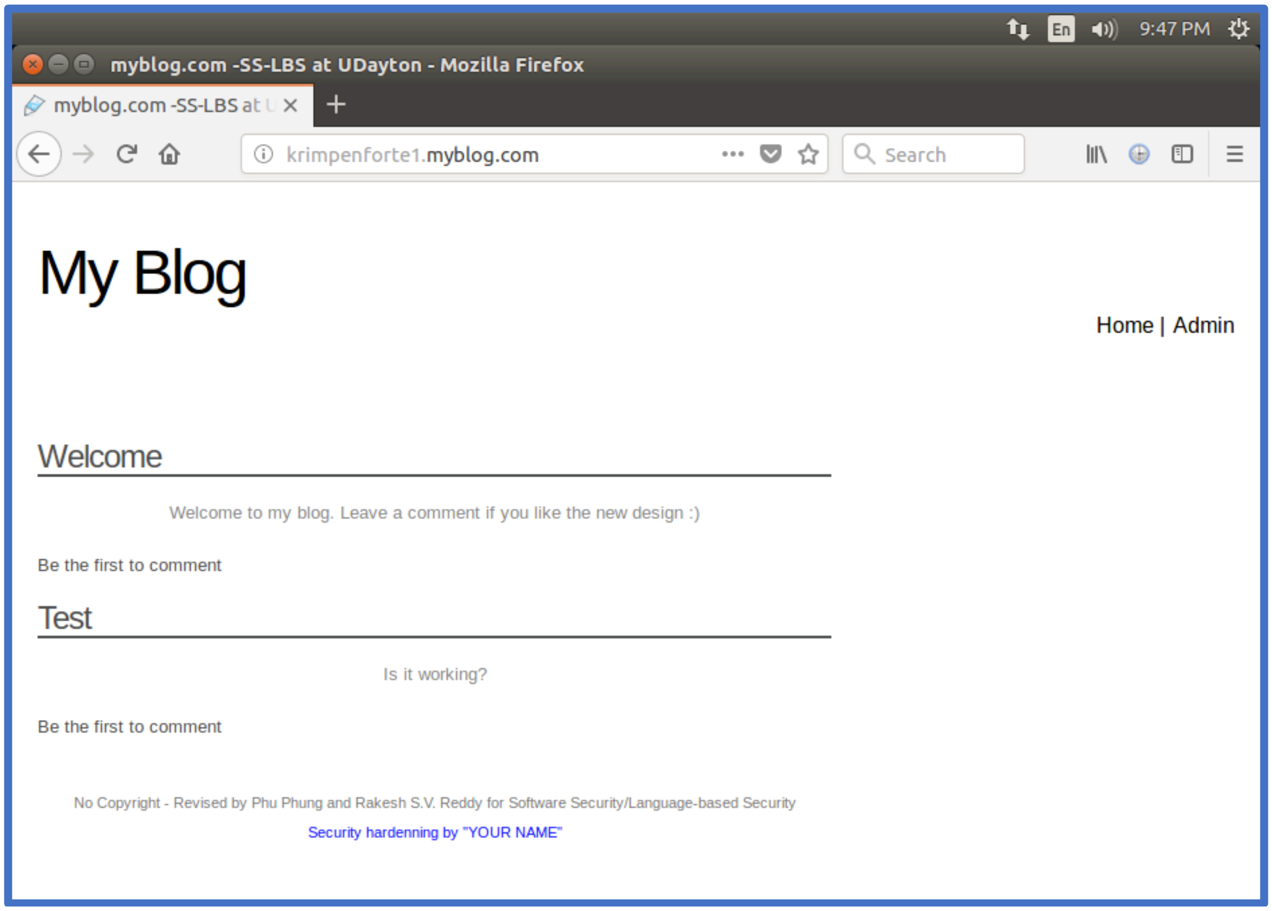
No Copyright - Revised by Phu Phung and Rakesh S.V. Reddy for Software Security/Language-based Security

Security hardenning by "YOUR NAME"

Figure 3: http://192.168.56.101/www.myblog.com/

iv. With local domain name

1. SEEDVM



My Blog

Home | Admin

Welcome

Welcome to my blog. Leave a comment if you like the new design :)

Be the first to comment

Test

Is it working?

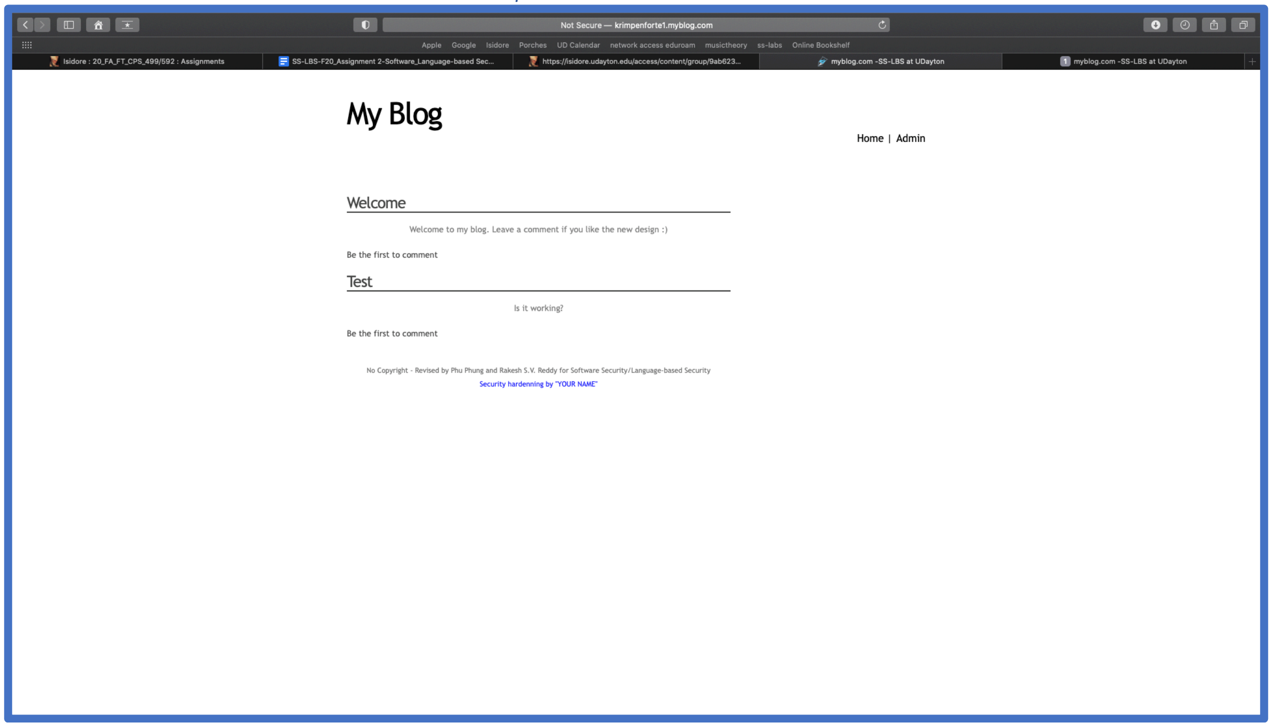
Be the first to comment

No Copyright - Revised by Phu Phung and Rakesh S.V. Reddy for Software Security/Language-based Security

Security hardenning by "YOUR NAME"

Figure 4: http://krimpenforte1.myblog.com/ on the VM

2. *Personal Computer*


Not Secure — krimpenforte1.myblog.com
Apple Google Isidore Porches UD Calendar network access eduroam musictheory ss-labs Online Bookshelf
Isidore : 20_FA_FT_CPS_499/592 : Assignments
SS-LBS-F20_Assignment 2-Software_Language-based Sec...
https://isidore.udayton.edu/access/content/group/9ab623...
myblog.com -SS-LBS at UDayton
myblog.com -SS-LBS at UDayton

My Blog

Home | Admin

Welcome

Welcome to my blog. Leave a comment if you like the new design :)

Be the first to comment

Test

Is it working?

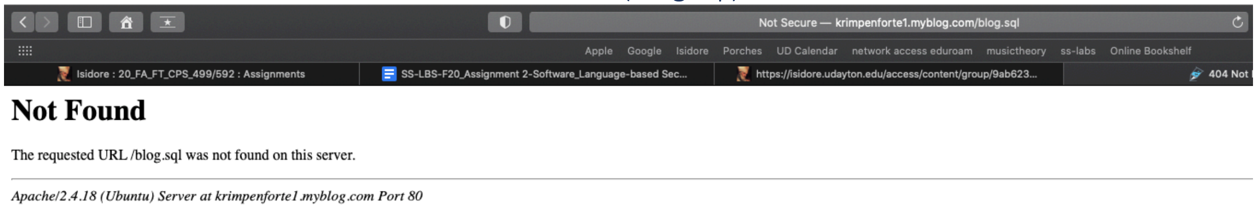
Be the first to comment

No Copyright - Revised by Phu Phung and Rakesh S.V. Reddy for Software Security/Language-based Security
Security hardenning by "YOUR NAME"

Figure 5: http://krimpenforte1.myblog.com/ on the Mac

c. Misconfiguration Security

v. Deleted the database file (blog.sql)


Not Secure — krimpenforte1.myblog.com/blog.sql
Apple Google Isidore Porches UD Calendar network access eduroam musictheory ss-labs Online Bookshelf
Isidore : 20_FA_FT_CPS_499/592 : Assignments
SS-LBS-F20_Assignment 2-Software_Language-based Sec...
https://isidore.udayton.edu/access/content/group/9ab623...
404 Not

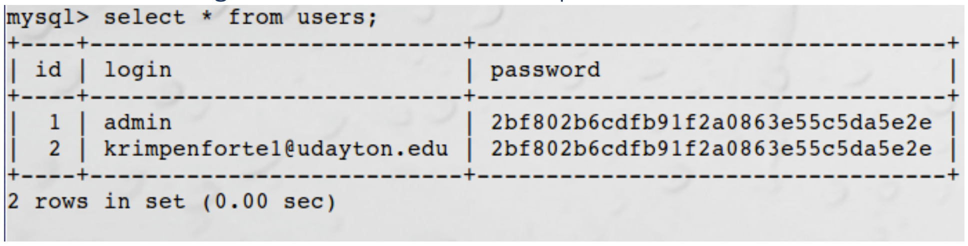
Not Found

The requested URL /blog.sql was not found on this server.

Apache/2.4.18 (Ubuntu) Server at krimpenforte1.myblog.com Port 80

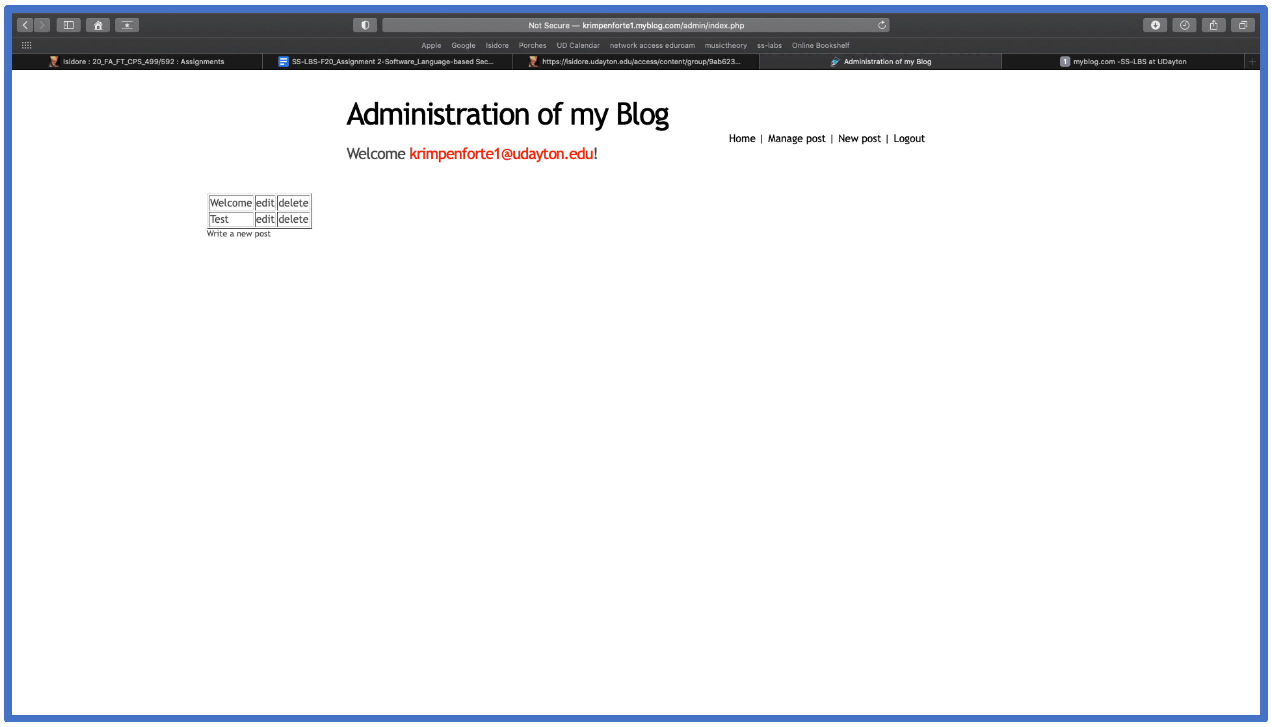
Figure 6: http://krimpenforte1.myblog.com/blog.sql not found

vi. Changed default username and password



```
mysql> select * from users;
+----+--------------------------+----------------------------------+
| id | login                    | password                         |
+----+--------------------------+----------------------------------+
|  1 | admin                    | 2bf802b6cdfb91f2a0863e55c5da5e2e |
|  2 | krimpenforte1@udayton.edu | 2bf802b6cdfb91f2a0863e55c5da5e2e |
+----+--------------------------+----------------------------------+
2 rows in set (0.00 sec)
```

Figure 7: Users available


Not Secure — krimpenforte1.myblog.com/admin/index.php
Apple Google Isidore Porches UD Calendar network access eduroam musictheory ss-labs Online Bookshelf
Isidore : 20_FA_FT_CPS_499/592 : Assignments
SS-LBS-F20_Assignment 2-Software_Language-based Sec...
https://isidore.udayton.edu/access/content/group/9ab623...
Administration of my Blog
myblog.com -SS-LBS at UDayton

Administration of my Blog

Welcome krimpenforte1@udayton.edu!

Home | Manage post | New post | Logout

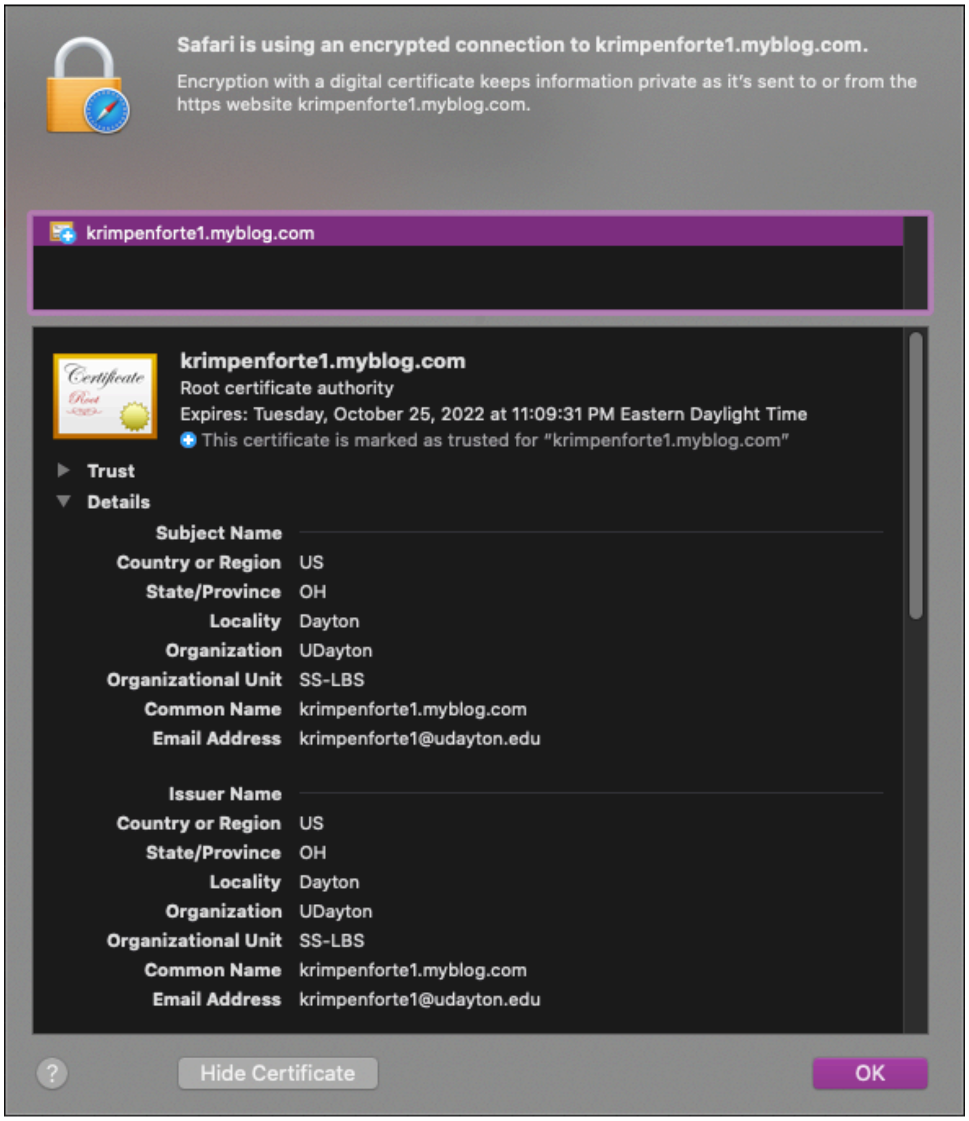
Welcome	edit	delete
Test	edit	delete

Write a new post

Figure 8: can log in with krimpenforte1@udayton.edu

d. HTTPS Setup

vii. Certificate was made



Safari is using an encrypted connection to krimpenforte1.myblog.com.

Encryption with a digital certificate keeps information private as it's sent to or from the https website krimpenforte1.myblog.com.

krimpenforte1.myblog.com

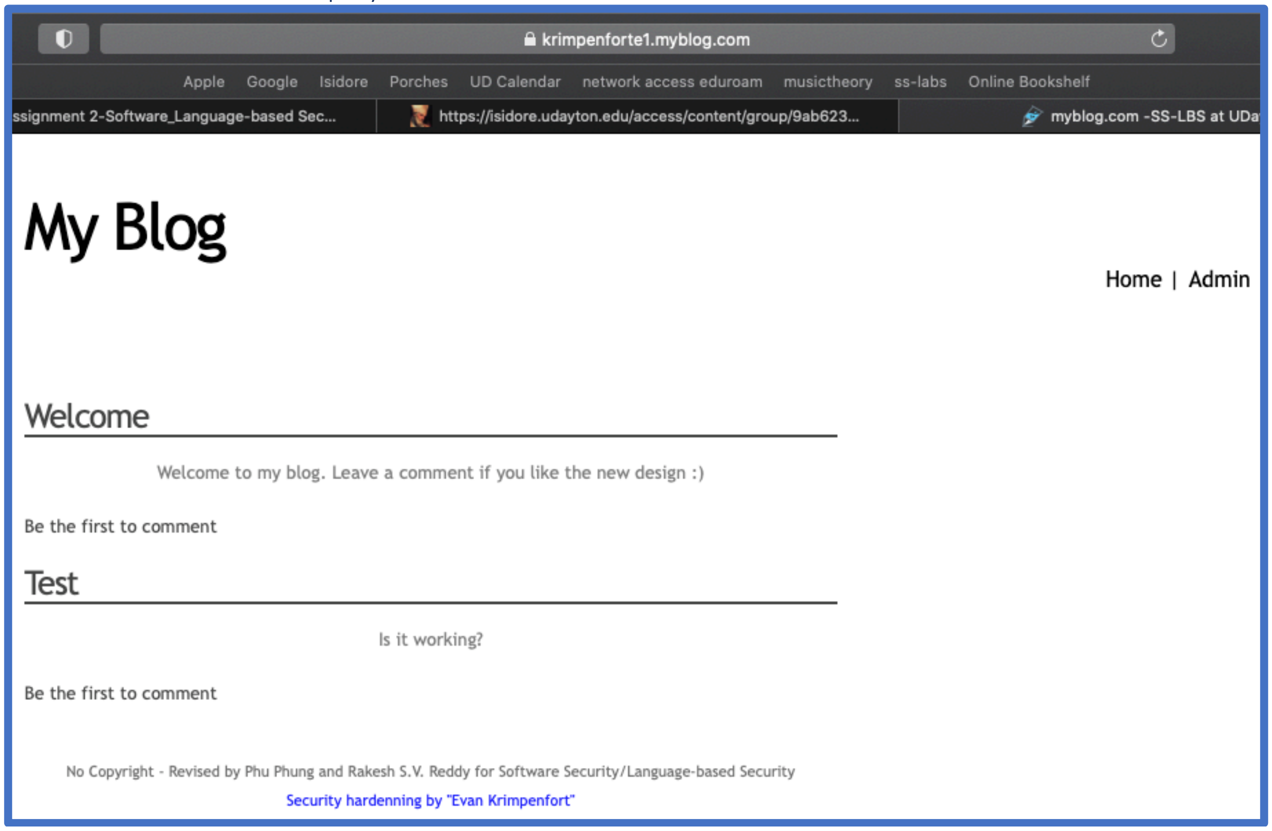
krimpenforte1.myblog.com
Root certificate authority
Expires: Tuesday, October 25, 2022 at 11:09:31 PM Eastern Daylight Time
This certificate is marked as trusted for "krimpenforte1.myblog.com"

▶ Trust
▼ Details

Subject Name	
Country or Region	US
State/Province	OH
Locality	Dayton
Organization	UDayton
Organizational Unit	SS-LBS
Common Name	krimpenforte1.myblog.com
Email Address	krimpenforte1@udayton.edu
Issuer Name	
Country or Region	US
State/Province	OH
Locality	Dayton
Organization	UDayton
Organizational Unit	SS-LBS
Common Name	krimpenforte1.myblog.com
Email Address	krimpenforte1@udayton.edu

Figure 9: Certificate Details

viii. Deployment was done


krimpenforte1.myblog.com
Apple Google Isidore Porches UD Calendar network access eduroam musictheory ss-labs Online Bookshelf
ssignment 2-Software_Language-based Sec...
https://isidore.udayton.edu/access/content/group/9ab623...
myblog.com -SS-LBS at UDa

My Blog

Home | Admin

Welcome

Welcome to my blog. Leave a comment if you like the new design :)

Be the first to comment

Test

Is it working?

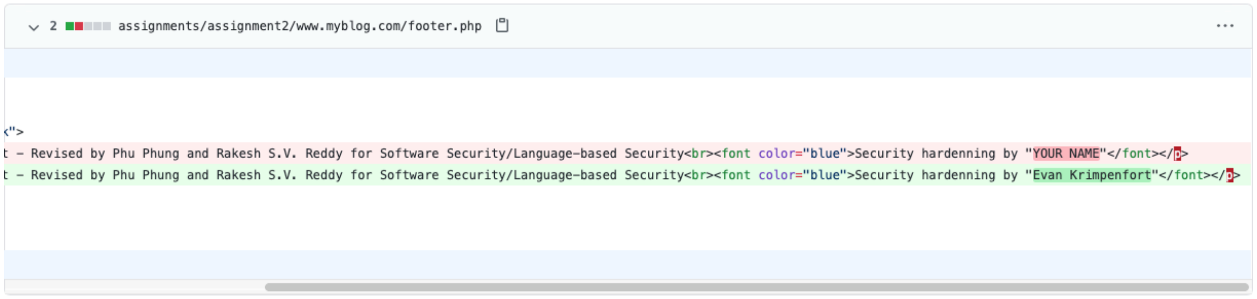
Be the first to comment

No Copyright - Revised by Phu Phung and Rakesh S.V. Reddy for Software Security/Language-based Security

Security hardening by "Evan Krimpenfort"

Figure 10: Site has https and the footer has been changed

e. Repository

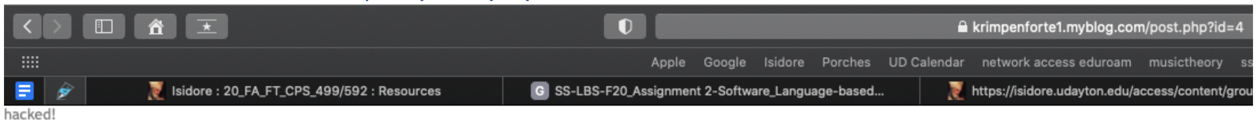


```
<">
t - Revised by Phu Phung and Rakesh S.V. Reddy for Software Security/Language-based Security<br><font color="blue">Security hardenning by "YOUR NAME"</font></p>
t - Revised by Phu Phung and Rakesh S.V. Reddy for Software Security/Language-based Security<br><font color="blue">Security hardenning by "Evan Krimpenfort"</font></p>
```

Figure 11: Footer was changed

Task I: Cross-Site Scripting (XSS) Prevention

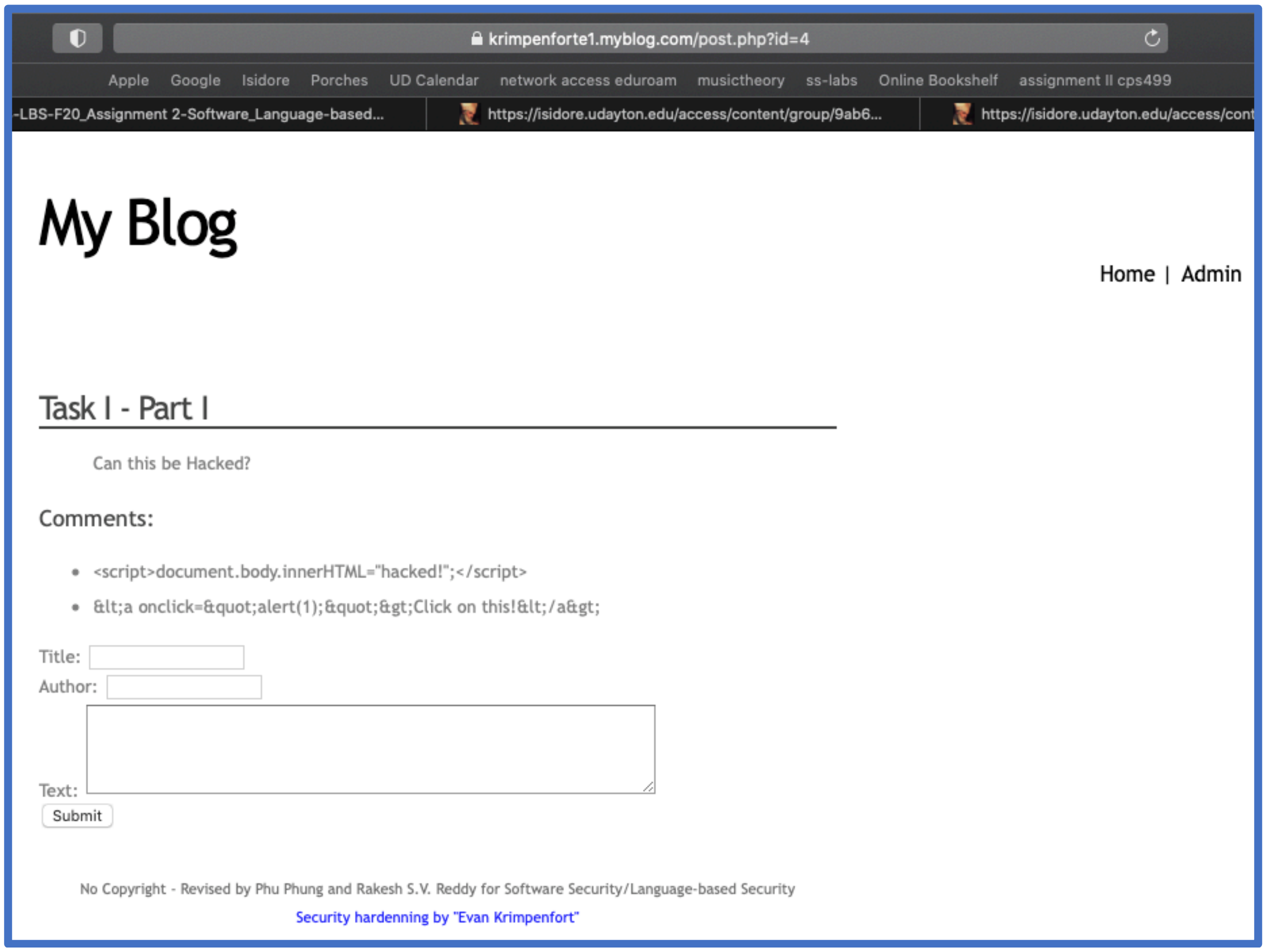
a. XSS vulnerability in **post.php** code


krimpenforte1.myblog.com/post.php?id=4
Apple Google Isidore Porches UD Calendar network access eduroam musictheory
Isidore : 20_FA_FT_CPS_499/592 : Resources
SS-LBS-F20_Assignment 2-Software_Language-based...
https://isidore.udayton.edu/access/content/grou

hacked!

Figure 12: hacked in *post.php?id=4*

By placing in the code *<script.document.body.innerHTML=hacked!";</script>*, whoever is to click on the post of *Task I – Part I*, they will be taken to a page that modifies the HTML to this text. We can fix the problem by going into our code of *classes/post.php* and encapsulate the *$comment->text* block of code (line 81). By doing this, we can see that in Figure 13 the first comment has been prevented of manipulation. Another fix that we can do is better help the function *htmlentities(..)* by encapsulating the inputs before they go into the database. We can do this by encapsulating those calls *$_POST[<insert name>]* with *htmlspecialchars(..)*. By doing this protection, we can see that in Figure 13 the second comment has been changed to show the specific html characters used.


krimpenforte1.myblog.com/post.php?id=4

My Blog

Home | Admin

Task I - Part I

Can this be Hacked?

Comments:

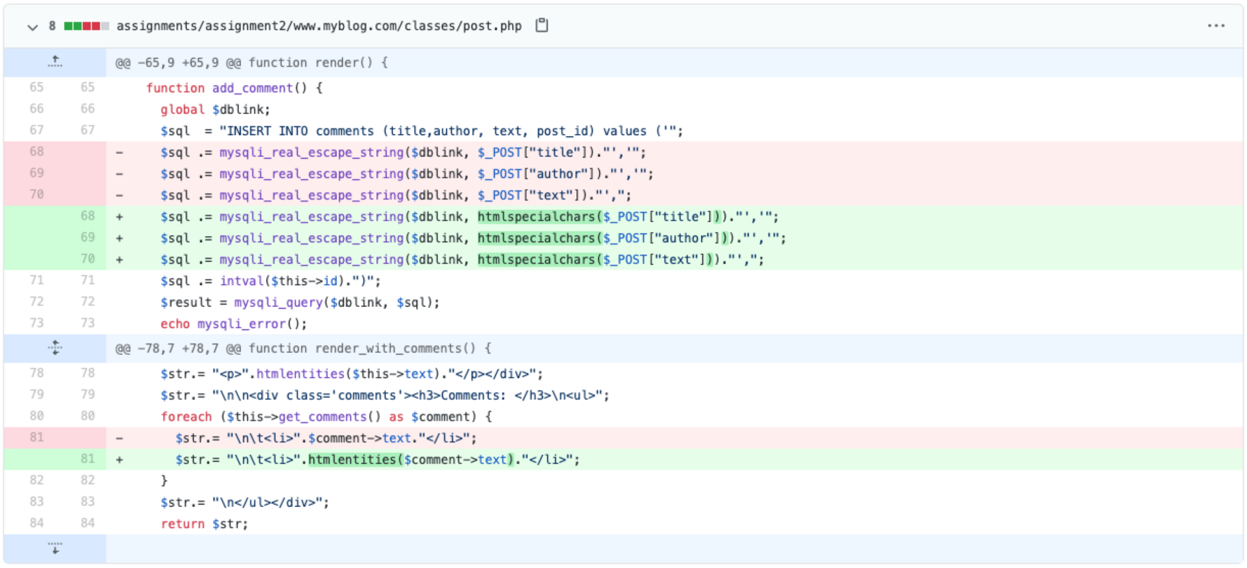
- <script>document.body.innerHTML="hacked!";</script>
- &lt;a onclick=&quot;alert(1);&quot;&gt;Click on this!&lt;/a&gt;

Title:
Author:
Text:
Submit

No Copyright - Revised by Phu Phung and Rakesh S.V. Reddy for Software Security/Language-based Security
Security hardenning by "Evan Krimpenfort"

Figure 13: fixed code with *htmlspecialchars* and *htmlentities*

By doing these fixes, we can safely say that the *classes/post.php* code has been protected from XSS scripting attacks. In Figure 14 the code is pushed and the changes are shown.



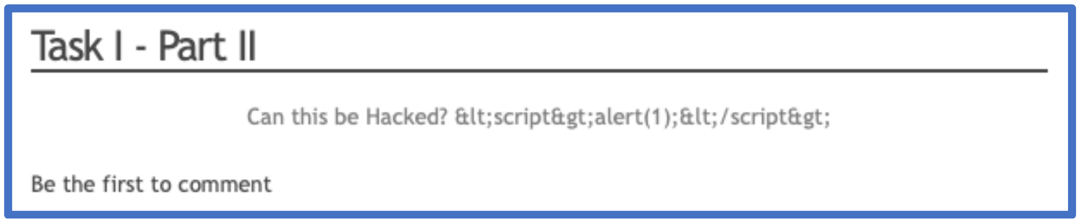
```
8 assignments/assignment2/www.myblog.com/classes/post.php
@@ -65,9 +65,9 @@ function render() {
   function add_comment() {
     global $dblink;
     $sql  = "INSERT INTO comments (title,author, text, post_id) values ('";
-    $sql .= mysqli_real_escape_string($dblink, $_POST["title"])."','";
-    $sql .= mysqli_real_escape_string($dblink, $_POST["author"])."','";
-    $sql .= mysqli_real_escape_string($dblink, $_POST["text"])."',";
+    $sql .= mysqli_real_escape_string($dblink, htmlspecialchars($_POST["title"]))."','";
+    $sql .= mysqli_real_escape_string($dblink, htmlspecialchars($_POST["author"]))."','";
+    $sql .= mysqli_real_escape_string($dblink, htmlspecialchars($_POST["text"]))."',";
     $sql .= intval($this->id).")";
     $result = mysqli_query($dblink, $sql);
     echo mysqli_error();
@@ -78,7 +78,7 @@ function render_with_comments() {
     $str.= "<p>".htmlentities($this->text)."</p></div>";
     $str.= "\n\n<div class='comments'><h3>Comments: </h3>\n<ul>";
     foreach ($this->get_comments() as $comment) {
-      $str.= "\n\t<li>".$comment->text."</li>";
+      $str.= "\n\t<li>".htmlentities($comment->text)."</li>";
     }
     $str.= "\n</ul></div>";
     return $str;
```

Figure 14: Fixed code

However, with these protections put in place, are there other areas where the code can be protected? Part b will show this.

b. Further Revisions

Another location that allows someone to post malicious code could be inside of the admin page. But, *index.php* is being protected by *classes/phpfix.php* with the function *h(..)*. The only further locations with revisions that could be made were further in *classes/post.php*. They were inside the *update(..)* and *create()* functions. In Figure 15, a new post shows the fix.



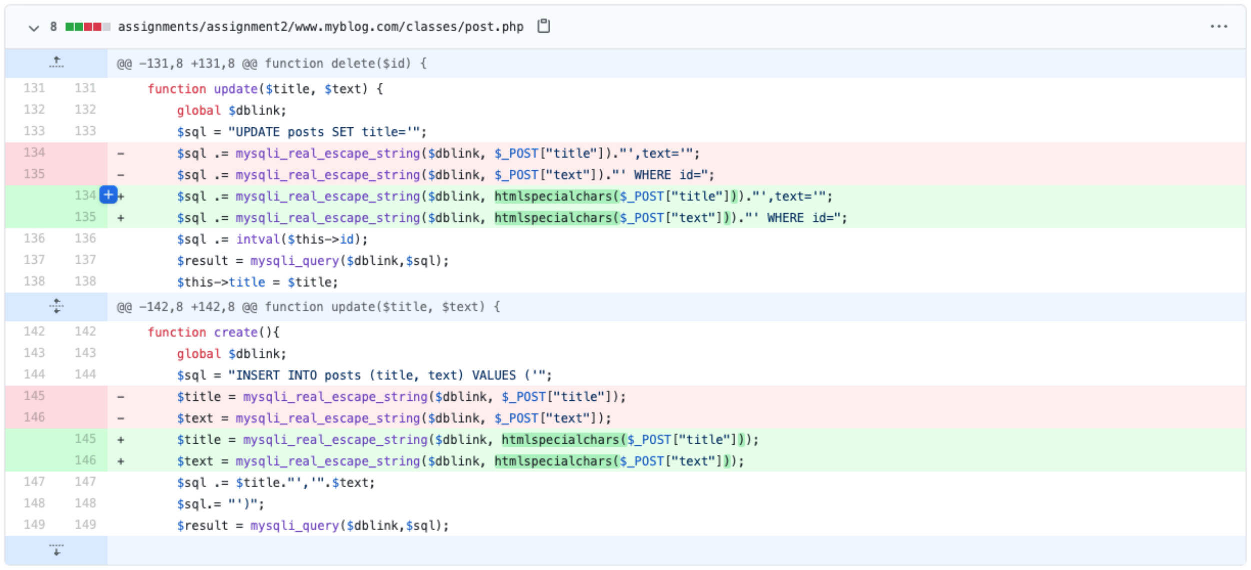
Task I - Part II

Can this be Hacked? <script>alert(1);</script>

Be the first to comment

Figure 15: Further Fix made for the actual post itself

The encapsulation used in each function was the *htmlspecialchars(..)* function for further sanitation. However, that was all I could find. The update to the repo can be found in Figure 16.

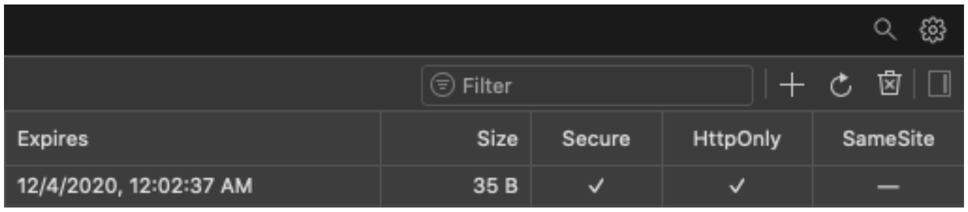


```
8 assignments/assignment2/www.myblog.com/classes/post.php
@@ -131,8 +131,8 @@ function delete($id) {
    function update($title, $text) {
        global $dblink;
        $sql = "UPDATE posts SET title='";
-       $sql .= mysqli_real_escape_string($dblink, $_POST["title"])."',text='";
-       $sql .= mysqli_real_escape_string($dblink, $_POST["text"])."' WHERE id=";
+       $sql .= mysqli_real_escape_string($dblink, htmlspecialchars($_POST["title"]))."',text='";
+       $sql .= mysqli_real_escape_string($dblink, htmlspecialchars($_POST["text"]))."' WHERE id=";
        $sql .= intval($this->id);
        $result = mysqli_query($dblink,$sql);
        $this->title = $title;
@@ -142,8 +142,8 @@ function update($title, $text) {
    function create(){
        global $dblink;
        $sql = "INSERT INTO posts (title, text) VALUES ('";
-       $title = mysqli_real_escape_string($dblink, $_POST["title"]);
-       $text = mysqli_real_escape_string($dblink, $_POST["text"]);
+       $title = mysqli_real_escape_string($dblink, htmlspecialchars($_POST["title"]));
+       $text = mysqli_real_escape_string($dblink, htmlspecialchars($_POST["text"]));
        $sql .= $title."','".$text;
        $sql.= "')";
        $result = mysqli_query($dblink,$sql);
```

Figure 16: Further fixed code

Task II: Session Hijacking Prevention

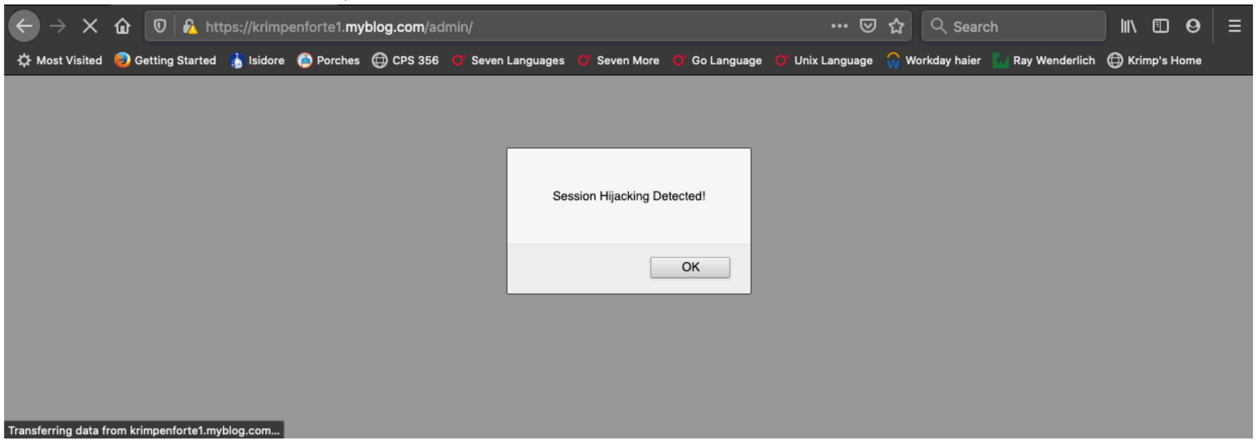
a. Secure the Cookies



Expires	Size	Secure	HttpOnly	SameSite
12/4/2020, 12:02:37 AM	35 B	✓	✓	—

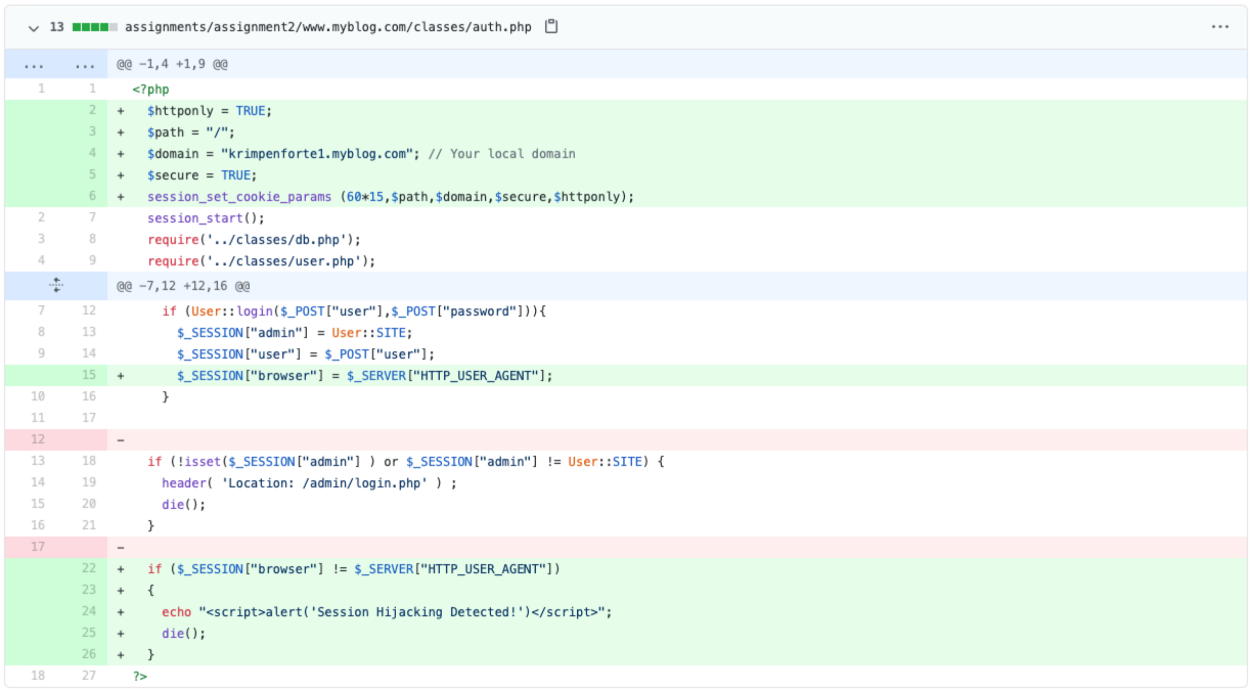
Figure 17: Secured the cookies

b. Defense in Depth



Session Hijacking Detected!

Figure 18: Session Hijacking Prevented



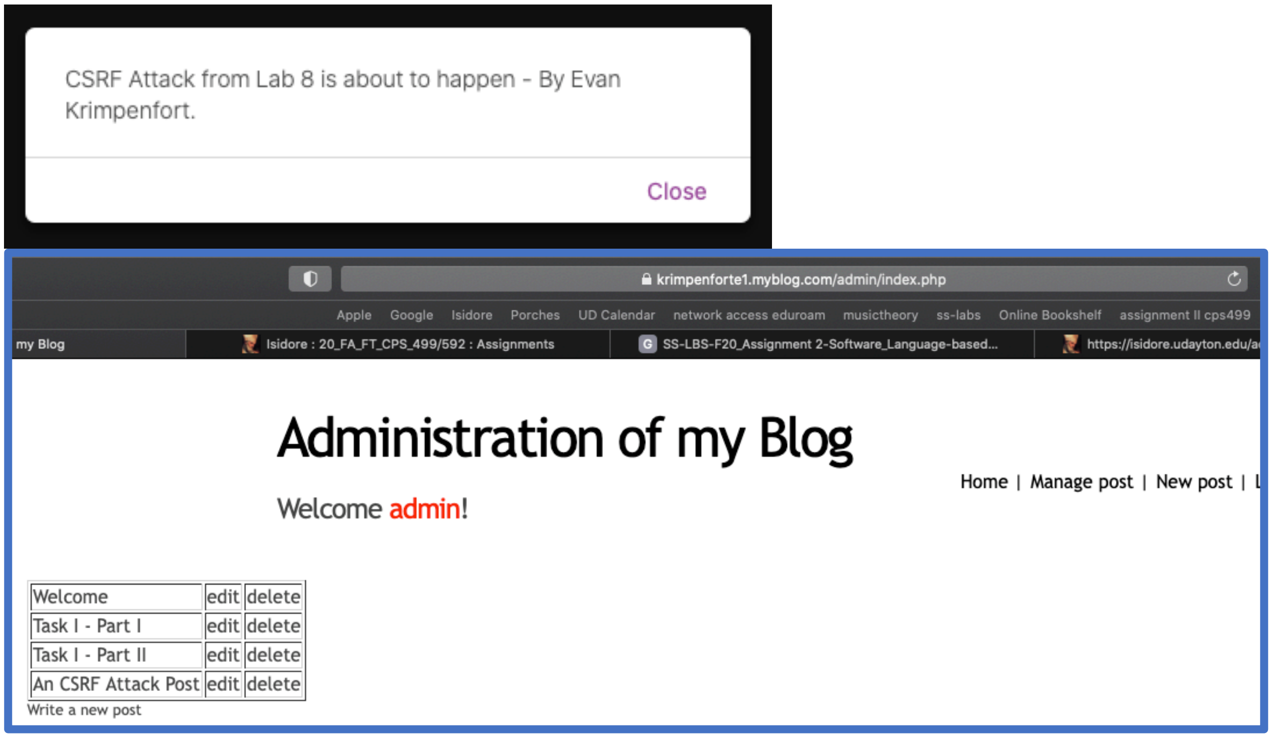
```
13 assignments/assignment2/www.myblog.com/classes/auth.php
@@ -1,4 +1,9 @@
  <?php
+  $httponly = TRUE;
+  $path = "/";
+  $domain = "krimpenforte1.myblog.com"; // Your local domain
+  $secure = TRUE;
+  session_set_cookie_params (60*15,$path,$domain,$secure,$httponly);
   session_start();
   require('../classes/db.php');
   require('../classes/user.php');
@@ -7,12 +12,16 @@
     if (User::login($_POST["user"],$_POST["password"])){
       $_SESSION["admin"] = User::SITE;
       $_SESSION["user"] = $_POST["user"];
+      $_SESSION["browser"] = $_SERVER["HTTP_USER_AGENT"];
     }

-
   if (!isset($_SESSION["admin"] ) or $_SESSION["admin"] != User::SITE) {
     header( 'Location: /admin/login.php' ) ;
     die();
   }
-
+  if ($_SESSION["browser"] != $_SERVER["HTTP_USER_AGENT"])
+  {
+    echo "<script>alert('Session Hijacking Detected!')</script>";
+    die();
+  }
  ?>
```

Figure 19: Additions for Session Hijacking Prevention

Task III: CSRF Prevention

a. CSRF vulnerability in the code of **admin/new.php**

This attack is able to happen because someone with administrational access logged in and they were able to use their access against them to post something. As we see in Figure 20, a post was made because the logged in user went to the http://192.168.56.101/csrflab.html page.



CSRF Attack from Lab 8 is about to happen - By Evan Krimpenfort.

Close
krimpenforte1.myblog.com/admin/index.php
Apple Google Isidore Porches UD Calendar network access eduroam musictheory ss-labs Online Bookshelf assignment II cps499
my Blog
Isidore : 20_FA_FT_CPS_499/592 : Assignments
SS-LBS-F20_Assignment 2-Software_Language-based...
https://isidore.udayton.edu/a

Administration of my Blog

Welcome admin!

Home | Manage post | New post |

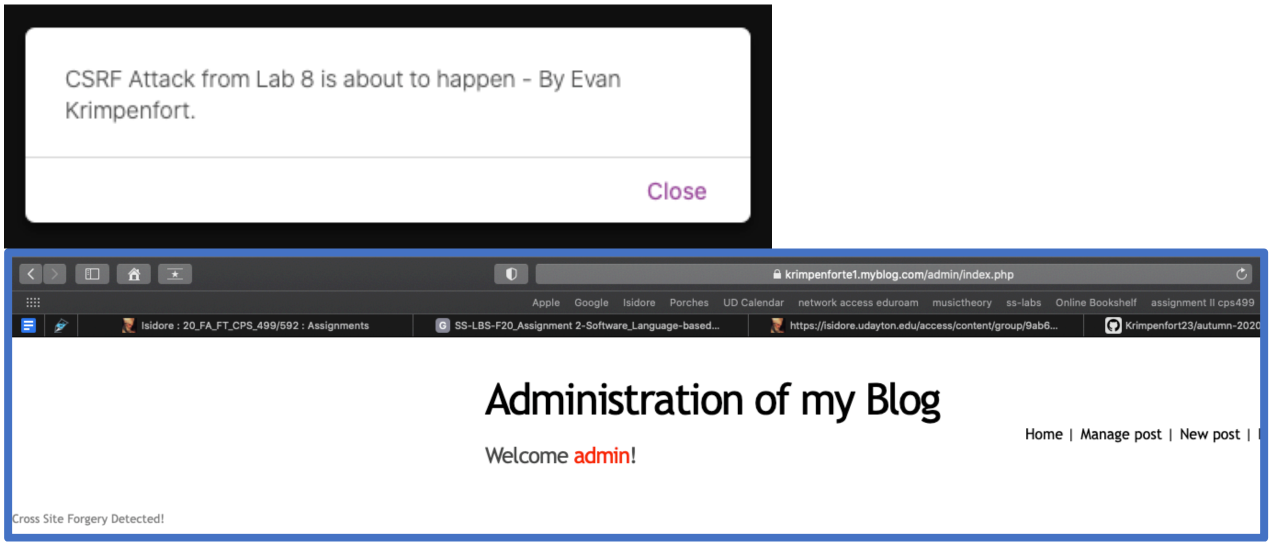
Welcome	edit	delete
Task I - Part I	edit	delete
Task I - Part II	edit	delete
An CSRF Attack Post	edit	delete

Write a new post

Figure 20: CSRF Attack

This attack is prevented by implementing a Secret Validation Token. We could have implemented other methods previously discussed in class like the Referrer Validation methodology or the Custom HTTP Header methodology, but the Secret Validation Token seemed quickest to implement and enough for the point of defending against a CSRF Attack.

The Secret Validation Token involves working in *admin/new.php* and *admin/index.php*. While inside of *admin/new.php*, a random variable is made and stored in the session. While inside of *admin/index.php*, that random variable is retrieved during the attack and is checked to see if *$nocsrftoken* is even set or if it matches. If the variable is not set or doesn't match, there is a CSRF attack taking place and that will trigger the site. The demonstration can be viewed in Figure 21.


CSRF Attack from Lab 8 is about to happen - By Evan Krimpenfort.

Close
krimpenforte1.myblog.com/admin/index.php
Apple Google Isidore Porches UD Calendar network access eduroam musictheory ss-labs Online Bookshelf assignment II cps499
Isidore : 20_FA_FT_CPS_499/592 : Assignments
SS-LBS-F20_Assignment 2-Software_Language-based...
https://isidore.udayton.edu/access/content/group/9ab6...
Krimpenfort23/autumn-2020
Home | Manage post | New post |

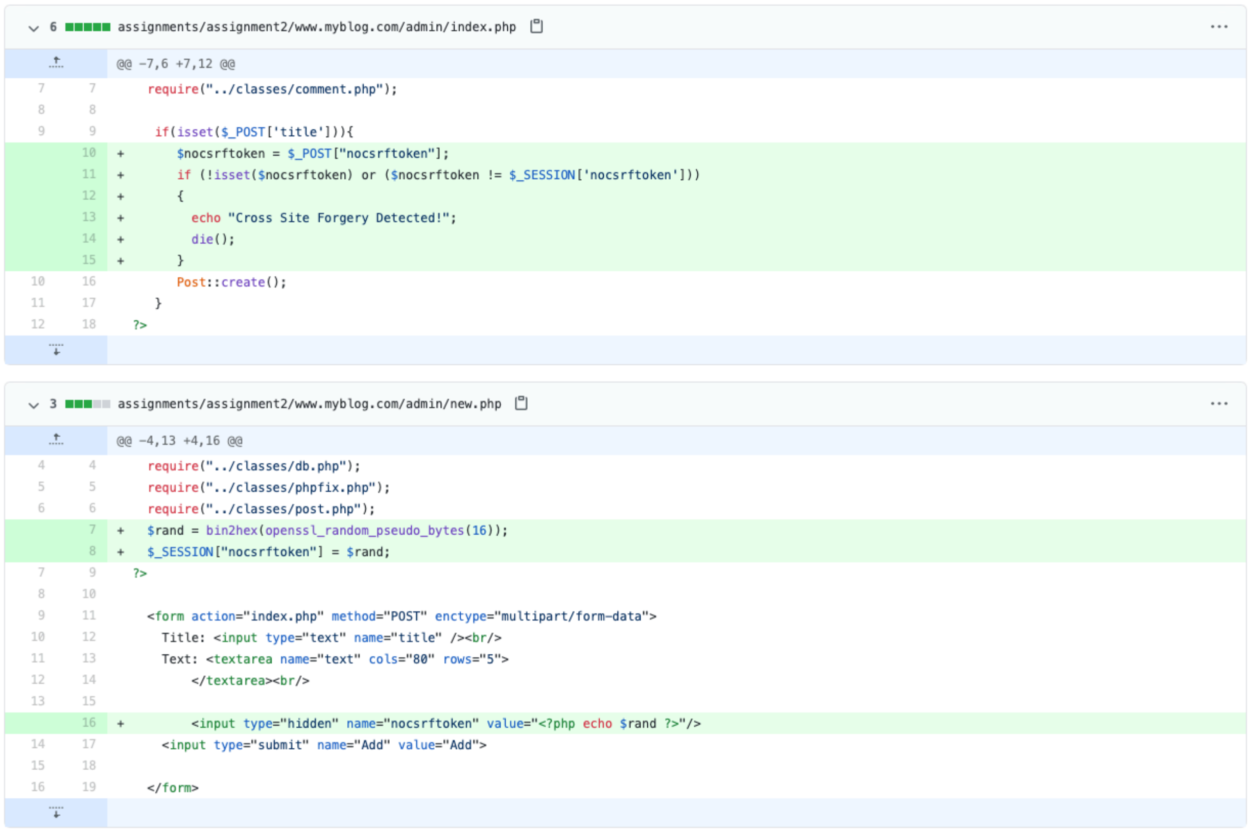
Administration of my Blog

Welcome admin!

Cross Site Forgery Detected!

Figure 21: CSRF Prevention

The code that was developed during this prevention can be viewed in Figure 22.

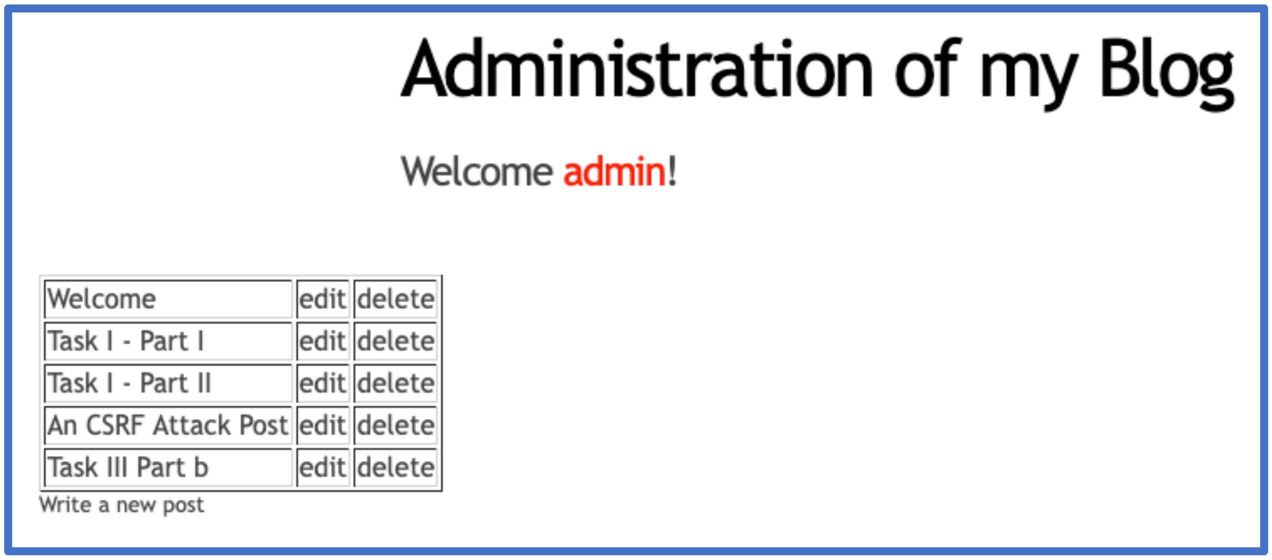

```
6 assignments/assignment2/www.myblog.com/admin/index.php
@@ -7,6 +7,12 @@
    require("../classes/comment.php");

     if(isset($_POST['title'])){
+        $nocsrftoken = $_POST["nocsrftoken"];
+        if (!isset($nocsrftoken) or ($nocsrftoken != $_SESSION['nocsrftoken']))
+        {
+          echo "Cross Site Forgery Detected!";
+          die();
+        }
        Post::create();
     }
  ?>
```

```
3 assignments/assignment2/www.myblog.com/admin/new.php
@@ -4,13 +4,16 @@
    require("../classes/db.php");
    require("../classes/phpfix.php");
    require("../classes/post.php");
+   $rand = bin2hex(openssl_random_pseudo_bytes(16));
+   $_SESSION["nocsrftoken"] = $rand;
  ?>

    <form action="index.php" method="POST" enctype="multipart/form-data">
      Title: <input type="text" name="title" /><br/>
      Text: <textarea name="text" cols="80" rows="5">
          </textarea><br/>

+         <input type="hidden" name="nocsrftoken" value="<?php echo $rand ?>"/>
      <input type="submit" name="Add" value="Add">

    </form>
```

Figure 22: Fixed Code for CSRF Attack

b. Further Revisions

Further Revisions could be found in both the *admin/edit.php*, *admin/index* and *admin/del.php* files. These scripts could allow CSRF attacks to happen. Therefore, additions were in need of making. Edit was just like new in that it was a *$_POST* so the same randomization and check in the index file could be done. However, with delete being a *$*_GET call, the randomization had to be placed in the index file, sent via the URL, and finally checked inside of the del file. By making these changes, the CSRF attacks could be stopped. All of those adjustments can be seen in Figure 27.



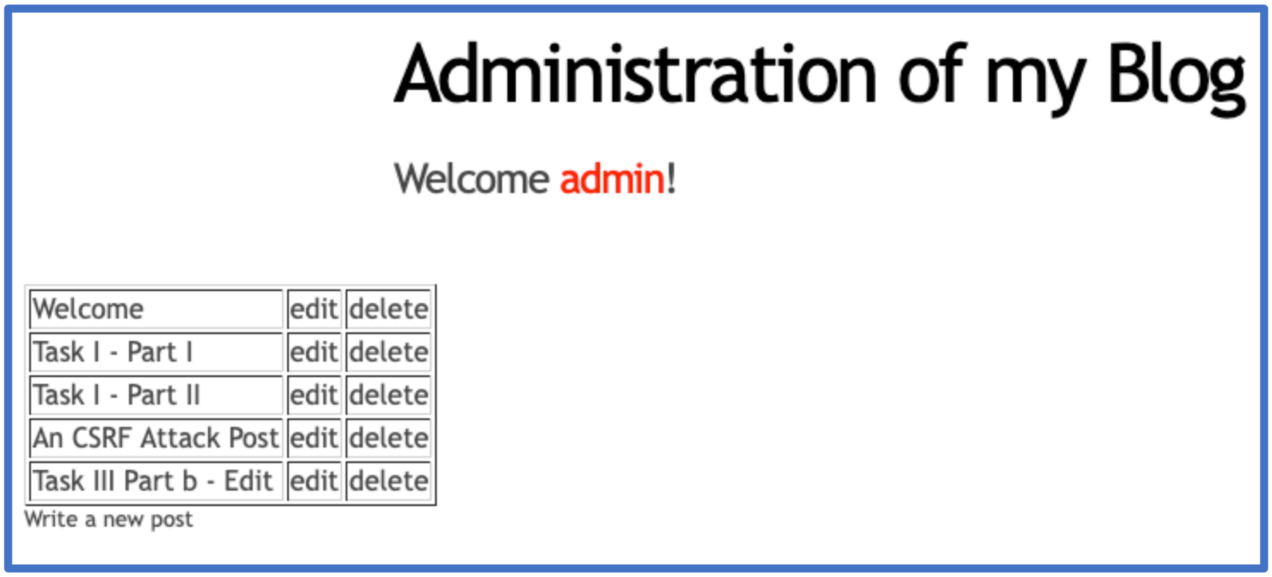
Administration of my Blog

Welcome admin!

Welcome	edit	delete
Task I - Part I	edit	delete
Task I - Part II	edit	delete
An CSRF Attack Post	edit	delete
Task III Part b	edit	delete

Write a new post

Figure 23: Successful add of a post



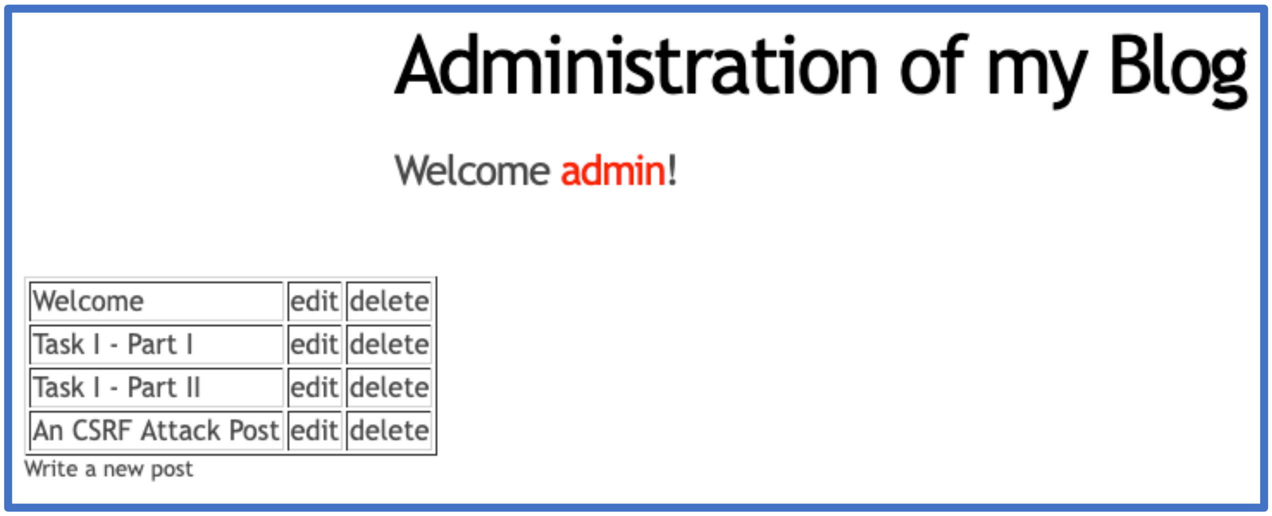
Administration of my Blog

Welcome admin!

Welcome	edit	delete
Task I - Part I	edit	delete
Task I - Part II	edit	delete
An CSRF Attack Post	edit	delete
Task III Part b - Edit	edit	delete

Write a new post

Figure 24: Successful edit of a post

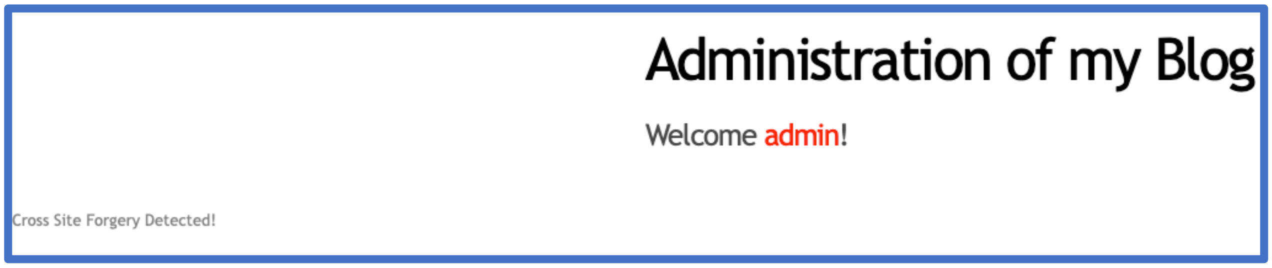

Administration of my Blog

Welcome admin!

Welcome	edit	delete
Task I - Part I	edit	delete
Task I - Part II	edit	delete
An CSRF Attack Post	edit	delete

Write a new post

Figure 25: Successful Deletion of a post

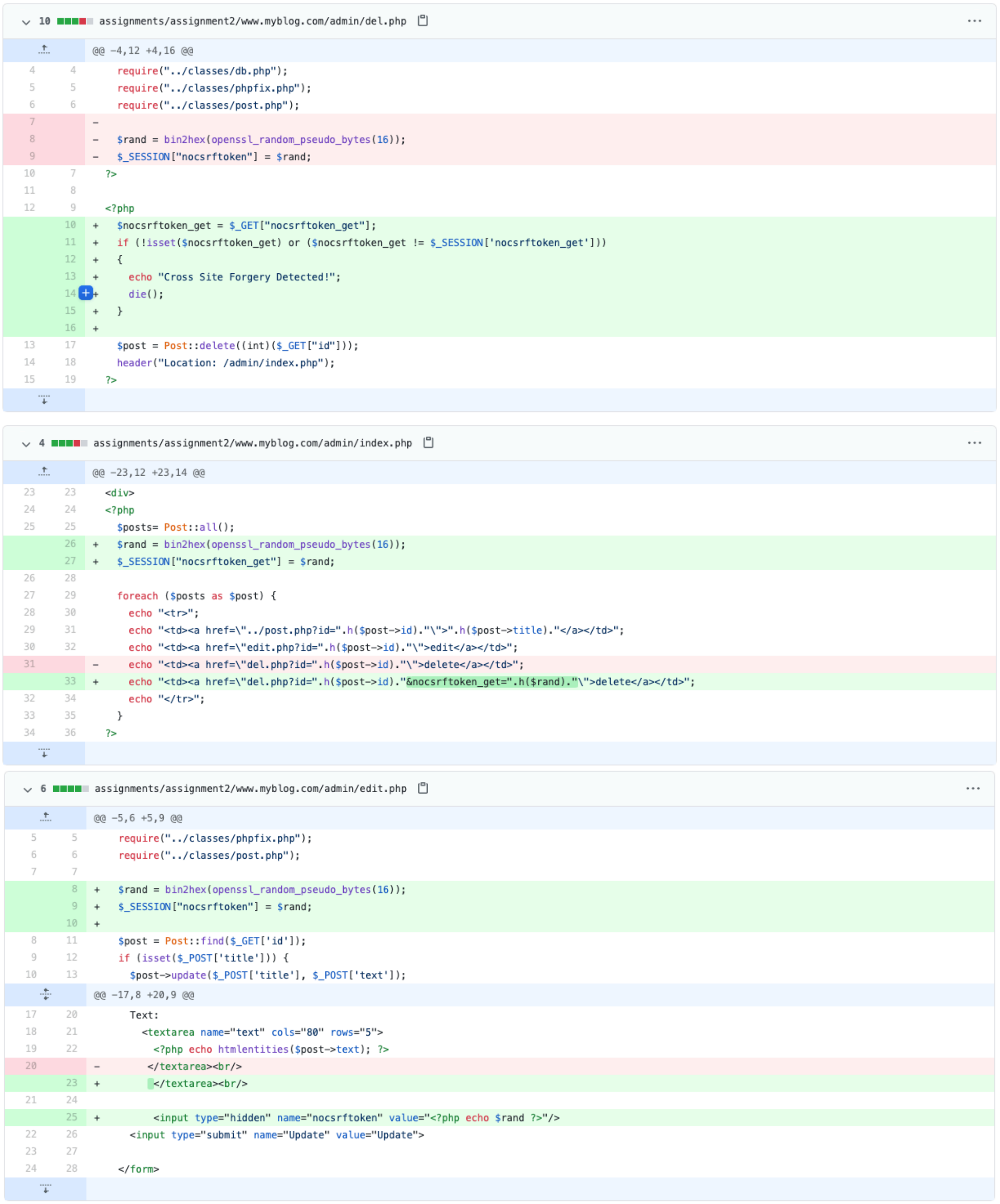

Administration of my Blog

Welcome admin!

Cross Site Forgery Detected!

Figure 26: Still Successful Protection Against CSRF

All changes made regard a successful admin account experience.



```
10 assignments/assignment2/www.myblog.com/admin/del.php
@@ -4,12 +4,16 @@
    require("../classes/db.php");
    require("../classes/phpfix.php");
    require("../classes/post.php");
-
-   $rand = bin2hex(openssl_random_pseudo_bytes(16));
-   $_SESSION["nocsrftoken"] = $rand;
  ?>

  <?php
+   $nocsrftoken_get = $_GET["nocsrftoken_get"];
+   if (!isset($nocsrftoken_get) or ($nocsrftoken_get != $_SESSION['nocsrftoken_get']))
+   {
+     echo "Cross Site Forgery Detected!";
+     die();
+   }
+
    $post = Post::delete((int)($_GET["id"]));
    header("Location: /admin/index.php");
  ?>
```

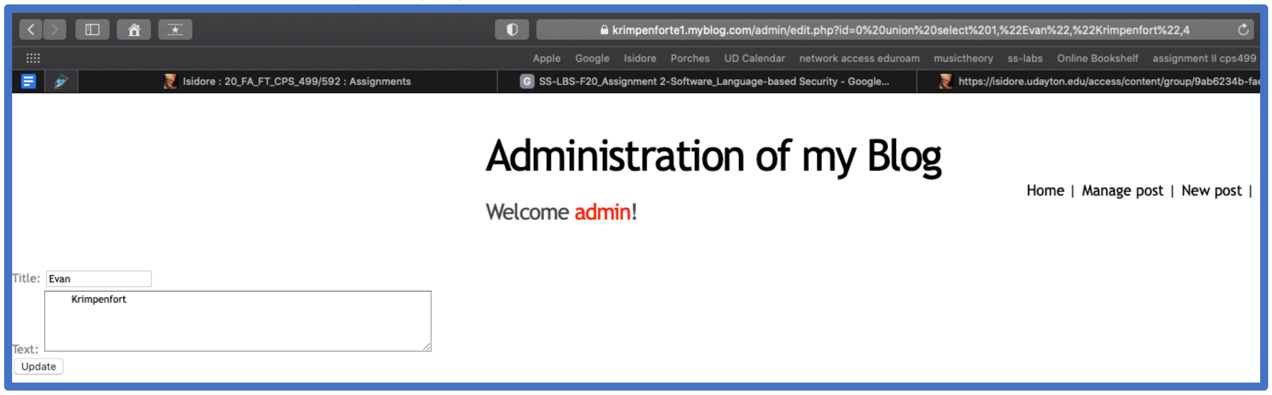
```
4 assignments/assignment2/www.myblog.com/admin/index.php
@@ -23,12 +23,14 @@
  <div>
  <?php
    $posts= Post::all();
+   $rand = bin2hex(openssl_random_pseudo_bytes(16));
+   $_SESSION["nocsrftoken_get"] = $rand;

    foreach ($posts as $post) {
      echo "<tr>";
      echo "<td><a href=\"../post.php?id=".h($post->id)."\">".h($post->title)."</a></td>";
      echo "<td><a href=\"edit.php?id=".h($post->id)."\">edit</a></td>";
-     echo "<td><a href=\"del.php?id=".h($post->id)."\">delete</a></td>";
+     echo "<td><a href=\"del.php?id=".h($post->id)."&nocsrftoken_get=".h($rand)."\">delete</a></td>";
      echo "</tr>";
    }
  ?>
```

```
6 assignments/assignment2/www.myblog.com/admin/edit.php
@@ -5,6 +5,9 @@
    require("../classes/phpfix.php");
    require("../classes/post.php");

+   $rand = bin2hex(openssl_random_pseudo_bytes(16));
+   $_SESSION["nocsrftoken"] = $rand;
+
    $post = Post::find($_GET['id']);
    if (isset($_POST['title'])) {
      $post->update($_POST['title'], $_POST['text']);
@@ -17,8 +20,9 @@
      Text:
        <textarea name="text" cols="80" rows="5">
          <?php echo htmlentities($post->text); ?>
-         </textarea><br/>
+          </textarea><br/>

+         <input type="hidden" name="nocsrftoken" value="<?php echo $rand ?>"/>
      <input type="submit" name="Update" value="Update">

    </form>
```

Figure 27: Code Revisions Seen in del.php, index.php and edit.php

Task IV: SQL Injection Prevention

a. SQLi in **admin/post.php**


Apple
Google
Isidore
Porches
UD Calendar
network access eduroam
musictheory
ss-labs
Online Bookshelf
assignment II cps499
Isidore : 20_FA_FT_CPS_499/592 : Assignments
SS-LBS-F20_Assignment 2-Software_Language-based Security - Google...
https://isidore.udayton.edu/access/content/group/9ab6234b-fa

```
krimpenforte1.myblog.com/admin/edit.php?id=0%20union%20select%201,%22Evan%22,%22Krimpenfort%22,4
```

Administration of my Blog

Welcome admin!

Home | Manage post | New post |

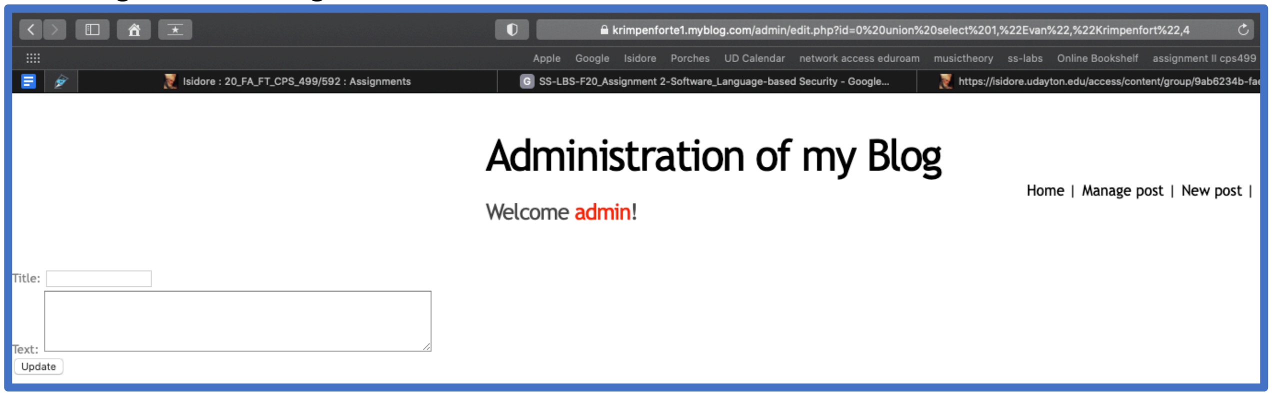
Title: Evan

Text: Krimpenfort

Update

Figure 28: Execution of the SQLi Attack

The reason this attack can happen in Figure 28 is because *id* is not protected inside of admin. The user asking doing this attack is not validated. Therefore, this needs to be protected by validating the user doing this attack.


Apple
Google
Isidore
Porches
UD Calendar
network access eduroam
musictheory
ss-labs
Online Bookshelf
assignment II cps499
Isidore : 20_FA_FT_CPS_499/592 : Assignments
SS-LBS-F20_Assignment 2-Software_Language-based Security - Google...
https://isidore.udayton.edu/access/content/group/9ab6234b-fa

```
krimpenforte1.myblog.com/admin/edit.php?id=0%20union%20select%201,%22Evan%22,%22Krimpenfort%22,4
```

Administration of my Blog

Welcome admin!

Home | Manage post | New post |

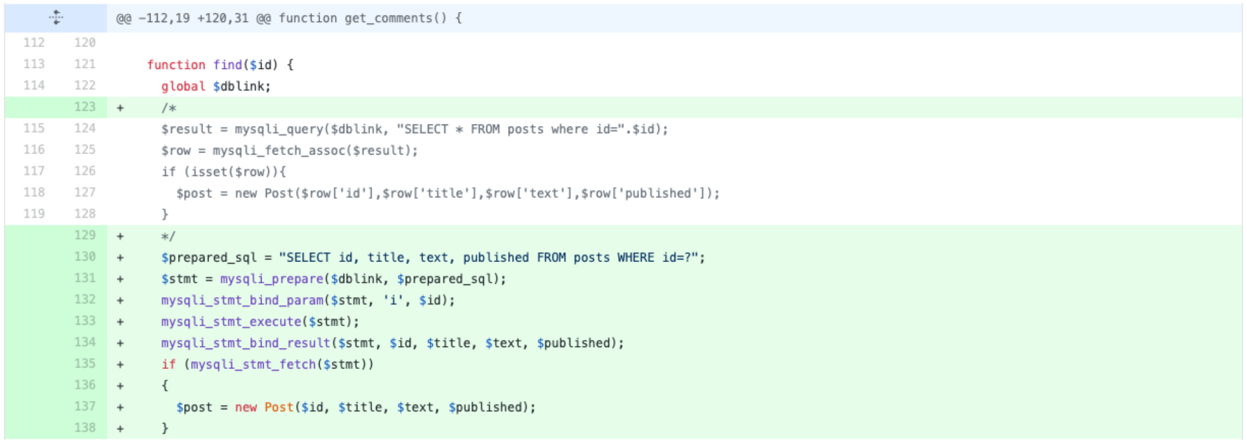
Title:

Text:

Update

Figure 29: SQLi Prevention

The prevention demonstrated in Figure 29 was done by disabling the ability to give user input in the first place. Union should not work now. By doing a *$prepared_sql*, update can no longer allow union to work. The code for that is shown in Figure 30.

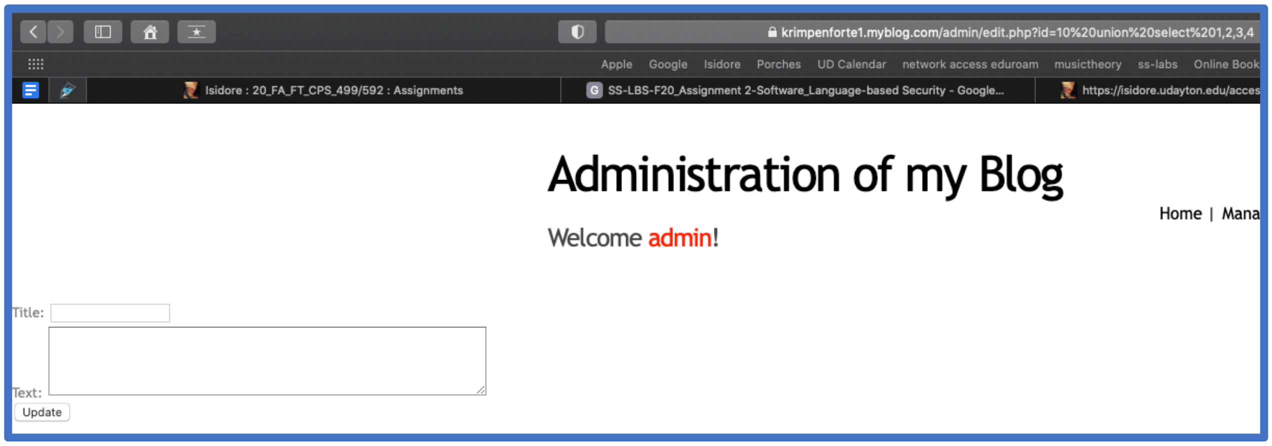


```
@@ -112,19 +120,31 @@ function get_comments() {

    function find($id) {
      global $dblink;
+     /*
      $result = mysqli_query($dblink, "SELECT * FROM posts where id=".$id);
      $row = mysqli_fetch_assoc($result);
      if (isset($row)){
        $post = new Post($row['id'],$row['title'],$row['text'],$row['published']);
      }
+     */
+     $prepared_sql = "SELECT id, title, text, published FROM posts WHERE id=?";
+     $stmt = mysqli_prepare($dblink, $prepared_sql);
+     mysqli_stmt_bind_param($stmt, 'i', $id);
+     mysqli_stmt_execute($stmt);
+     mysqli_stmt_bind_result($stmt, $id, $title, $text, $published);
+     if (mysqli_stmt_fetch($stmt))
+     {
+       $post = new Post($id, $title, $text, $published);
+     }
```

Figure 30: Code Revisions in Update(..) for SQLi

b. Further Revisions

In Figure 30, a demonstration on editing a post that doesn't exist doesn't cause an error and union was not able to post 2 and 3, thus keeping the SQLi Prevention.


krimpenforte1.myblog.com/admin/edit.php?id=10%20union%20select%201,2,3,4
Apple Google Isidore Porches UD Calendar network access eduroam musictheory ss-labs Online Book
Isidore : 20_FA_FT_CPS_499/592 : Assignments
SS-LBS-F20_Assignment 2-Software_Language-based Security - Google...
https://isidore.udayton.edu/acces

Administration of my Blog

Welcome admin!

Home | Mana

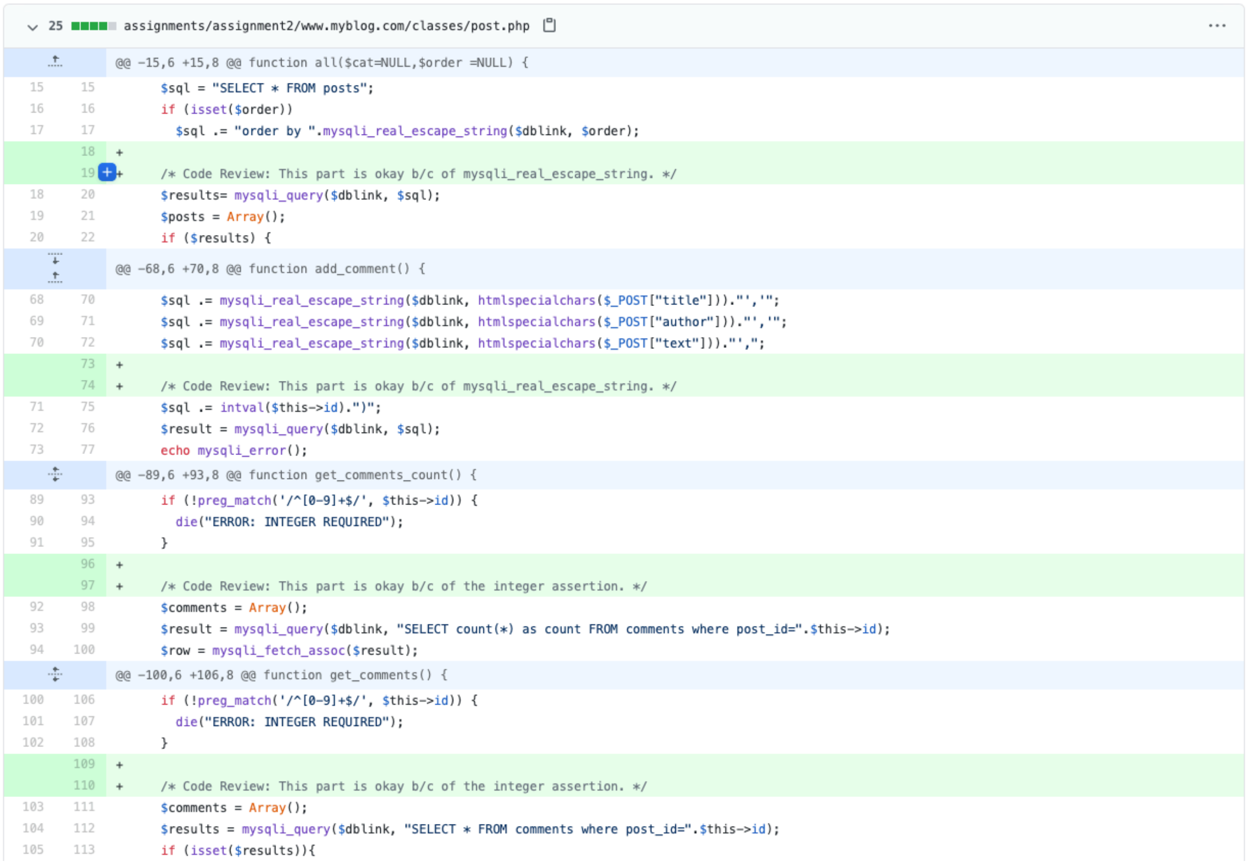
Title:

Text:

Update

Figure 31: Fixing edit through update by making sure it's null (which it is)

By going through all of *classes/post.php*, All methods besides the fix made all had protections. Whether that was integer assertion or escape protections. A snippet of this can be seen in Figure 32.



```
25 assignments/assignment2/www.myblog.com/classes/post.php
@@ -15,6 +15,8 @@ function all($cat=NULL,$order =NULL) {
        $sql = "SELECT * FROM posts";
        if (isset($order))
          $sql .= "order by ".mysqli_real_escape_string($dblink, $order);
+
+       /* Code Review: This part is okay b/c of mysqli_real_escape_string. */
        $results= mysqli_query($dblink, $sql);
        $posts = Array();
        if ($results) {
@@ -68,6 +70,8 @@ function add_comment() {
        $sql .= mysqli_real_escape_string($dblink, htmlspecialchars($_POST["title"]))."','";
        $sql .= mysqli_real_escape_string($dblink, htmlspecialchars($_POST["author"]))."','";
        $sql .= mysqli_real_escape_string($dblink, htmlspecialchars($_POST["text"]))."',";
+
+       /* Code Review: This part is okay b/c of mysqli_real_escape_string. */
        $sql .= intval($this->id).")";
        $result = mysqli_query($dblink, $sql);
        echo mysqli_error();
@@ -89,6 +93,8 @@ function get_comments_count() {
        if (!preg_match('/^[0-9]+$/', $this->id)) {
          die("ERROR: INTEGER REQUIRED");
        }
+
+       /* Code Review: This part is okay b/c of the integer assertion. */
        $comments = Array();
        $result = mysqli_query($dblink, "SELECT count(*) as count FROM comments where post_id=".$this->id);
        $row = mysqli_fetch_assoc($result);
@@ -100,6 +106,8 @@ function get_comments() {
        if (!preg_match('/^[0-9]+$/', $this->id)) {
          die("ERROR: INTEGER REQUIRED");
        }
+
+       /* Code Review: This part is okay b/c of the integer assertion. */
        $comments = Array();
        $results = mysqli_query($dblink, "SELECT * FROM comments where post_id=".$this->id);
        if (isset($results)){
```

Figure 32: Further Analysis Snippet