Course: PROG38263

Assignment: 3-4

Member 1 Name and Student ID:Trevor Clancy 991525079

Member 2 Name and Student ID: Aidan Hodgins 991524642

Section: 34777

Instructor: Syed Tanbeer

Vulnerabilities

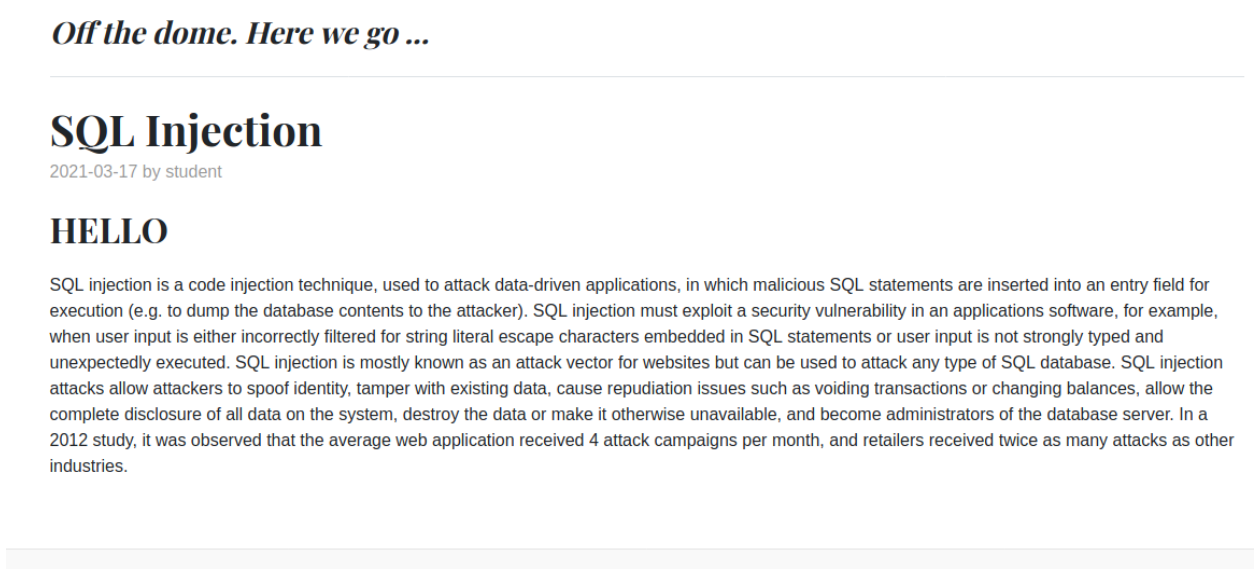
The application has the following known vulnerabilities that you must find and exploit.

V-1.

Cross-Site Scripting (XSS)

The website doesn’t check user inputs for any scripting languages and will allow for the execution of those scripting languages one the code reaches the users browser.

Before The Fix



After The Fix

Off the dome. Here we go ...

SQL Injection

2021-03-17 by student

<h1> HELLO </h1>SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker). SQL injection must exploit a security vulnerability in an applications software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database. SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server. In a 2012 study, it was observed that the average web application received 4 attack campaigns per month, and retailers received twice as many attacks as other industries.

The security implementation is extremely simple. Within editarticle.php we used the htmlspecialchars() function for the $content variable that will be used to display the content of the article. This function will find the html special characters and nullify them so that instead of being scripts that get executed they get converted into plaintext, that way the script never executes.

Implementation

```
if($_SERVER['REQUEST_METHOD'] == 'GET') {
        $aid = $_GET['aid'];
        $result=get_article($dbconn, $aid);
        $row = pg_fetch_array($result, 0);
} elseif ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $title = $_POST['title'];

        $str = $title;
        $pattern = "/(SELECT|FROM|WHERE|ALTER|TABLE|ADD|AND|AS|AVG|BETWEEN|CASE|THEN|ELSE|END|COUNT|C
        $value = preg_match($pattern, $str);
        $content = htmlspecialchars($_POST['content']);
        $aid = $_POST['aid'];

        if($value > 0) {
```

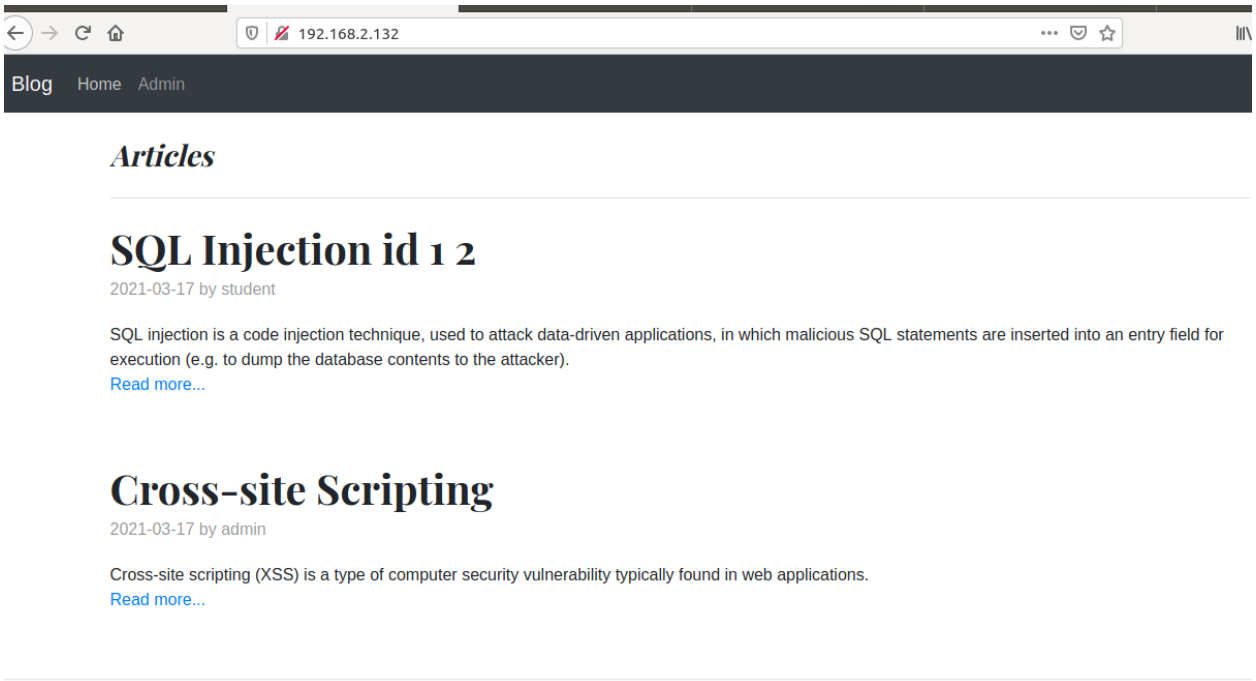
Open *Un... Save
Aidan Hodgins 991524642
Tab Width: 8 Ln 1, Col 24 INS

V-2.

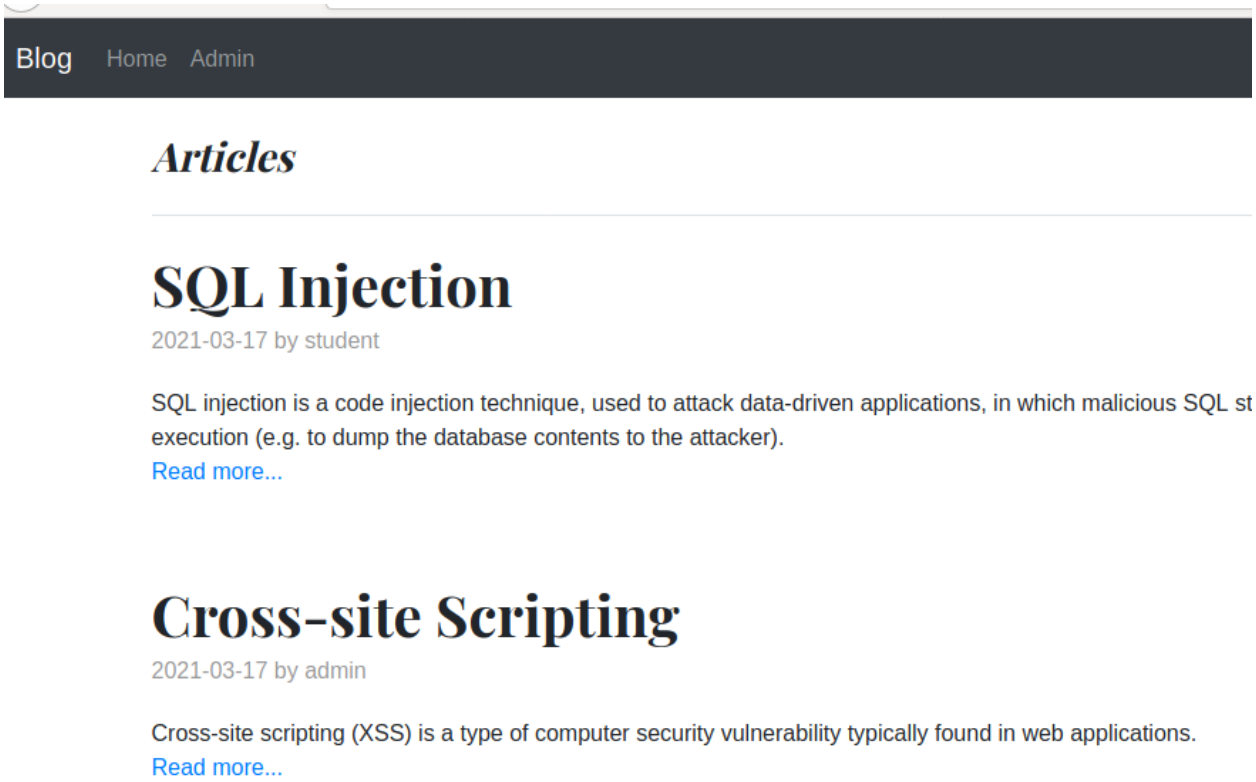
SQL Injection

SQL Injection can be exploited if the website doesn’t check for any SQL code inside the content boxes that the user provides, leading to information from the database leaking and being visible by the malicious actor.

Before Fix:



After Fix:



Implementation:

So implementing it I did a regex with a pattern checker, looking for every single key command used in SQL. If any one of the commands is found in the title for edit article, a print statement at the top of the page will be printed telling the user they can't put any SQL commands into the title, and will also stop the changes from being sent to the database, stopping any of the data leaking from the database.

```
if($_SERVER['REQUEST_METHOD'] == 'GET') {
        $aid = $_GET['aid'];
        $result=get_article($dbconn, $aid);
        $row = pg_fetch_array($result, 0);
} elseif ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $title = $_POST['title'];

        $str = $title;
        $pattern = "/(SELECT|FROM|WHERE|ALTER|TABLE|ADD|AND|AS|AVG|BETWEEN|CASE|THEN|ELSE|END|COUNT|CREATE|DELETE|(
        $value = preg_match($pattern, $str);
        $content = htmlspecialchars($_POST['content']);
        $aid = $_POST['aid'];

        if($value > 0) {

} else {

        $result=update_article($dbconn, $title, $content, $aid);
        }
        Header ("Location: /");
}
```

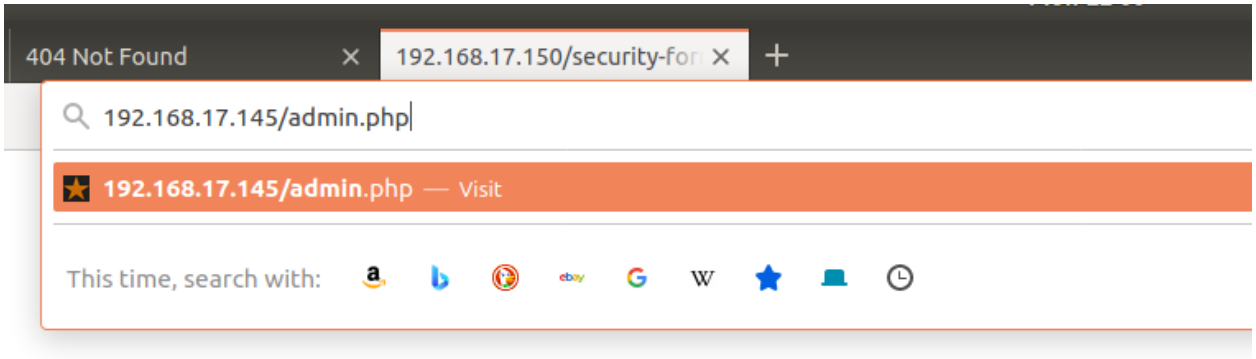
Open *Un... Save
Aidan Hodgins 991524642
Tab Width: 8 Ln 1, Col 24 INS

```
CREATE|DELETE|GROUP|BY|HAVING|JOIN|ON|INSERT|INTO|VALUES|IS|NULL|NOT|LIKE|LIMIT|MAX|MIN|OR|ORDER|LEFT|ROUND|DISTINCT|SUM|WITH)/";
```

Open *Un... Save
Aidan Hodgins 991524642
Tab Width: 8 Ln 1, Col 24 INS

V-3.

Broken Access Control (i.e. you can do things without authenticating that you should be able to do.)

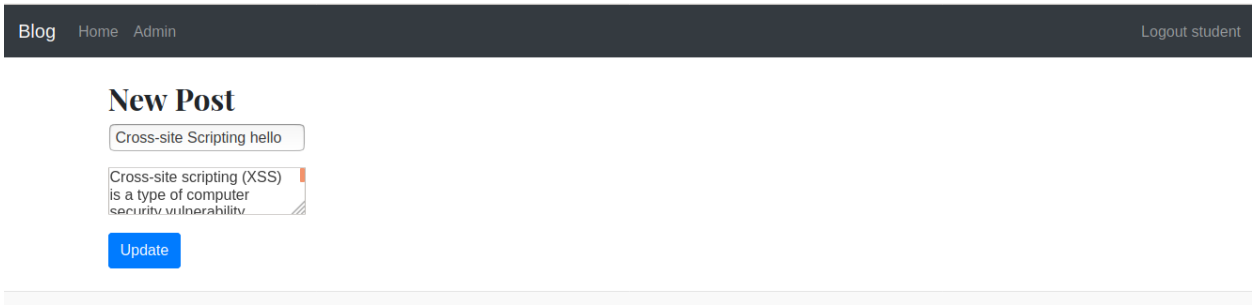


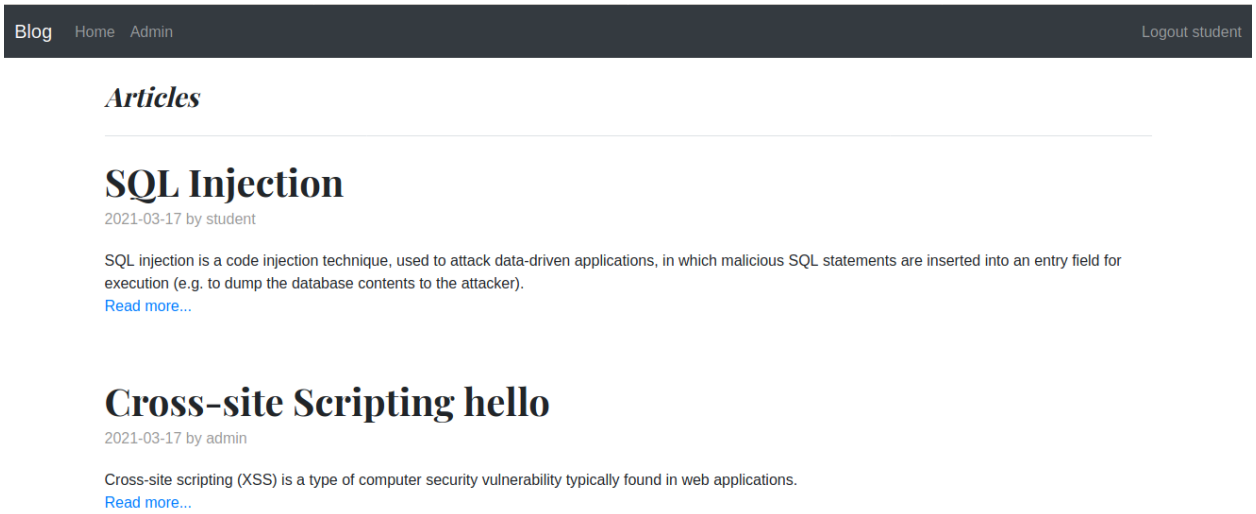
V-4.

Missing role-based access control enforcement and management (i.e. Blog authors should only be able to delete their posts. Admins can delete anyone's posts. There is currently no mechanism for changing or managing the roles for users. Actually, there is no mechanism for even creating or managing users without issuing manual SQL against the database).

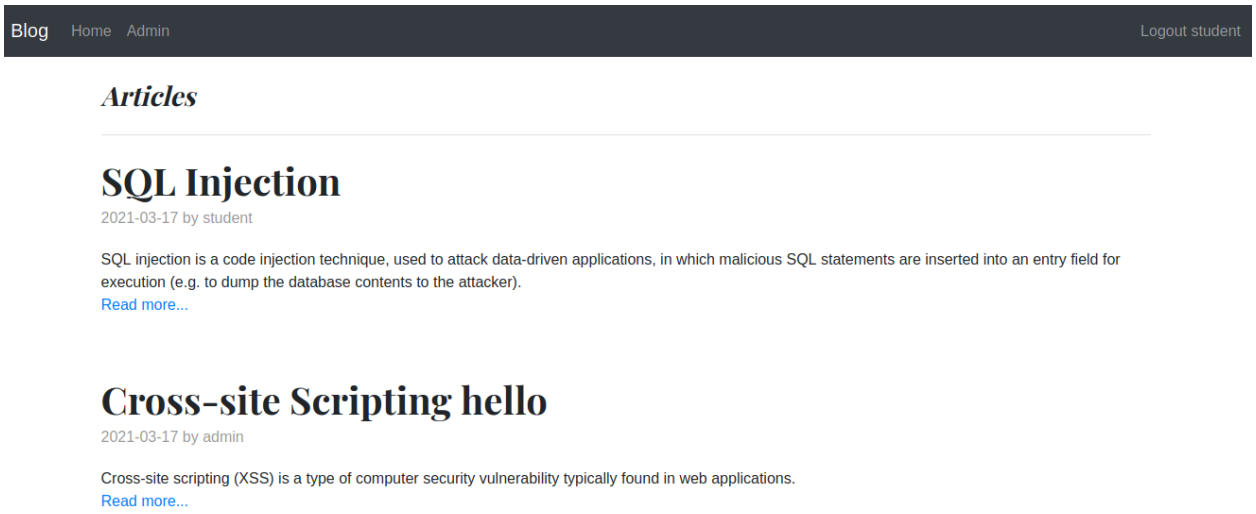
Improper implementation of Role-based access control can allow users to modify other users data while missing the proper powers to allow them to do so. And with this website student users have the power to modify files the admin created.

Before Fix


Blog
Home
Admin
Logout student
New Post
Cross-site Scripting hello
Cross-site scripting (XSS) is a type of computer
Update


Blog
Home
Admin
Logout student
Articles
SQL Injection
2021-03-17 by student
SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker).
Read more...
Cross-site Scripting hello
2021-03-17 by admin
Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications.
Read more...

After Fix:


Blog
Home
Admin
Logout student
Articles
SQL Injection
2021-03-17 by student
SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker).
Read more...
Cross-site Scripting hello
2021-03-17 by admin
Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications.
Read more...

Implementation:

In this implementation a check is performed after the user logs into the page. It checks to see what there user id is, if the id isn't equal to 1 and an admins article appears, the two links used to delete and modify the article become links redirecting the users to the index.php, stopping them from modifying the files.

```
<tr>
  <td><a href='article.php?aid=<?php echo $row['aid'] ?>'><?php echo $row['title'] ?></a></td>
  <td><?php echo $row['author'] ?></td>
  <td><?php echo substr($row['date'],0,10) ?></td>
        <?php if($_SESSION['id'] != 1 && $row['author'] == "admin") {?>
                <td><a href="/index.php"><i class="fa fa-pencil-square-o fa-2x" aria-hidden="true"></i></a></td>
                <td><a href="/index.php"><i class="fa fa-times fa-2x" aria-hidden="true"></i></a></td>
<?php } else {?>
        <td><a href="/editarticle.php?aid=<?php echo $row['aid'] ?>"><i class="fa fa-pencil-square-o fa-2x" aria-hidden="true"></i></a></td>
        <td><a href="/deletearticle.php?aid=<?php echo $row['aid'] ?>"><i class="fa fa-times fa-2x" aria-hidden="true"></i></a></td>
<?php }?>


</tr>
```

Open *Un... Save

Aidan Hodgins 991524642

V-5.

Insecure password handling and storage.

Handling Vulnerability: The passwords are not encrypted as they are inputted through the login.php page.This could result in man-in-the-middle attacks against the users and admin of the web

server. Especially the admin role, if this password is able to be obtained then all data can be deleted or modified.

Storage Vulnerability: User data is kept in the database unencrypted creating a vulnerability if an attacker was able to get to the database they can see all the passwords.

After Fix:

Select data Show structure Alter table New item

Select | Search | Sort | Limit 50 | Text length 100 | Action Select

```
SELECT * FROM "authors" LIMIT 50 (0.001 s) Edit
```

Modify	id	created_on	username	password	role
edit	1	2021-03-21 19:41:23.473865+00	admin	482c811da5d5b4bc6d497ffa98491e38	admin
edit	2	2021-03-21 19:41:23.478386+00	student	482c811da5d5b4bc6d497ffa98491e38	user

Whole result: 2 rows | Modify: Save | Selected (0): Edit Clone Delete | Export (2)

Import

Code Implemented:

Login.php: In the login.php page the password obtained from the form is received from the POST method and encrypted to authenticate the password stored in the database. This was the only update we needed to bring to this PHP file as we are only concerned on the process of encrypting passwords as they log into the website.

Updated_Code | Trevor_Aidan_A3-4 / Assignment-3-4 / code / **login.php** / Jump to

Clancy-Github Update login.php

1 contributor

72 lines (63 sloc) | 1.83 KB

```php
<?php include("templates/page_header.php");?>
<?php

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $result = authenticate_user($dbconn, $_POST['username'], md5($_POST['password']));
        if (pg_num_rows($result) == 1) {
                $_SESSION['username'] = $_POST['username'];
                $_SESSION['authenticated'] = True;
                $_SESSION['id'] = pg_fetch_array($result)['id'];
                //Redirect to admin area
                header("Location: /admin.php");
        }
}
```

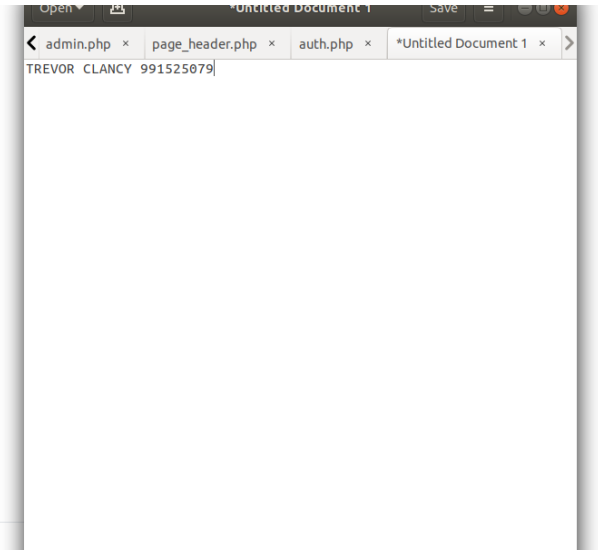
admin.php | page_header.php | auth.php | *Untitled Document 1

```
TREVOR CLANCY 991525079
```

Db.php: The authenticate_user() function in the db.php page is used to fix this vulnerability as we have the encrypted password stored in the database already being compared to the password given by the client logging in.

```php
function update_article($dbconn, $title, $content, $aid) {
        $query=
                "UPDATE articles
                SET
                title='$title',
                content='$content'
                WHERE
                aid='$aid'";
        return run_query($dbconn, $query);
}

function authenticate_user($dbconn, $username, $password) {
        $query=
                "SELECT
                authors.id as id,
                authors.username as username,
                md5(authors.password) as password,
                authors.role as role
                FROM
                authors
                WHERE
                username='".$_POST['username']."'
                AND
                password='".$_POST['password']."'
                LIMIT 1";
        return run_query($dbconn, $query);
}
?>
```



Other Potential Security Controls: There are many different hashing options for passwords such as SHA256 or SHA512, as well as many different ways of securing client data handling and storing passwords. A useful technique that could be used is the practice of salting where a randomly generated value is attached to the client password as it is encrypted in the database. Using functions available in PostgreSQL such as crypt(), and password_hash() both give the option for a customized salt as well as verification for analyzing the hash.

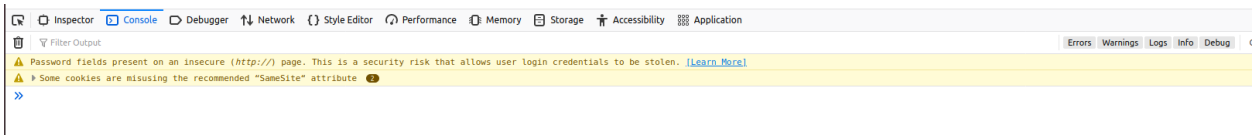
V-6.

CSRF

V-7.

The entire website, including the login page, is served over plaintext

Vulnerability: Having the whole website served over http instead of https leaves open the opportunity for attackers to use the cookies on the website to discover passwords and other key data involved in session hijacking.



HTTP.

V-8.The web application has no logging except for the default logs generated by Nginx.

Not implementing logging specific to your application can lead to potentially dangerous activities not being recorded properly or the format of logs can be hard to read if you aren't sure what you are looking for in the automated logs generated. To help alleviate this issue we implemented logging in login.php where we create logs

whenever an improper sign-in occurs and store the logs in a file called error.log

Implementation:

```php
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
        $result = authenticate_user($dbconn, $_POST['username'], $_POST['password']);
        if (pg_num_rows($result) == 1) {
                $_SESSION['username'] = $_POST['username'];
                $_SESSION['authenticated'] = True;
                $_SESSION['id'] = pg_fetch_array($result)['id'];

                //Redirect to admin area
                header("Location: /admin.php");
        } else {
                $date = new DateTime();
                $date = $date->format("y:m:d h:i:s");
                $log_file_data = $log_filename ." " . $date . $_POST['username'] ."\n";
                error_log($log_file_data, 3, "./error.log");
        }
}
```

Open *Un... Save

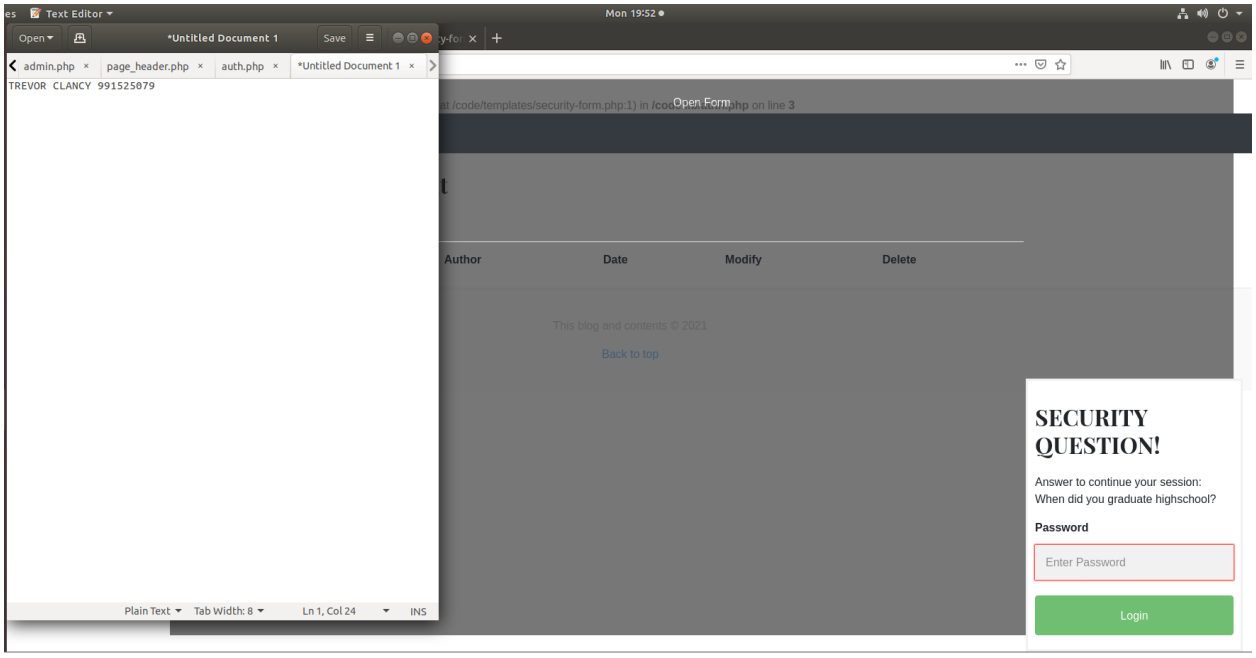
Aidan Hodgins 991524642

Tab Width: 8 Ln 1, Col 24 INS

V-9.

Authentication Vulnerability: The website only uses single-factor authentication. On the login.php page there is no multi-factor authentication in place to further authenticate to the users to the web server. Since the server just uses a single-factor authentication method it is extremely vulnerable potentially leading to a brute force attack. A password as simple as “password123” that both the admin and other user author has can be found and broken extremely easily.

Code Implemented: To fix this vulnerability, the security-form.php is included in the admin.php to add a further authentication factor for the user. This example is in the scenario when the admin logs in where the admin has a secret security question that they must answer to continue onto the page.



Security-form.php: This page has the purpose of acting as a layer of protection to the admin.php page having the client who logged in answer a security question to continue their session. Through the use of the function openForm() and closeForm() the security question can be opened and when answered correctly, will close the form of this page allowing the client to access the site.

Trevor_Aidan_A3-4/secu | 404 Not Found | Login

https://github.com/Clancy-Github/Trevor_Aidan_A3-4/blob/Updated_Code/Assignment-3-4/code/templ

110 lines (89 sloc) | 2.15 KB

```html
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
body {font-family: Arial, Helvetica, sans-serif;}
* {box-sizing: border-box;}

/* Button used to open the contact form - fixed at the bottom of the page */
.open-button {
  background-color: #555;
  color: white;
  padding: 16px 20px;
  border: none;
  cursor: pointer;
  opacity: 0.8;
  position: fixed;
  bottom: 23px;
  right: 28px;
  width: 1500px;
  height: 1500px;
}

/* The popup form - hidden by default */
.form-popup {
  display: none;
  position: fixed;
  bottom: 0;
  right: 15px;
  border: 3px solid #f1f1f1;
  z-index: 9;
}

/* Add styles to the form container */
.form-container {
  max-width: 300px;
  padding: 10px;
  background-color: white;
```

Open | *Untitled Document 1 | Save

admin.php | page_header.php | auth.php | *Untitled Document 1

```
TREVOR CLANCY 991525079
```

```html
  opacity: 1;
}
</style>
</head>
<body>



<button class="open-button" onclick="openForm()">Open Form</button>

<div class="form-popup" id="myForm">
  <form action="/action_page.php" class="form-container">
    <h1>SECURITY QUESTION!</h1>
    <p>Answer to continue your session: When did you graduate highschool?</p>

    <label for="psw"><b>Password</b></label>
    <input type="password" placeholder="Enter Password" name="psw" required>

    <button type="submit" class="btn">Login</button>

  </form>
</div>

<script>
function openForm() {
  document.getElementById("myForm").style.display = "block";
}

function closeForm() {
  document.getElementById("myForm").style.display = "none";
}
</script>

</body>
</html>
```

admin.php | page_header.php | auth.php | *Untitled Document 1

```
TREVOR CLANCY 991525079
```

Other Secure Options: Some of the other ways that we could have implemented multi-factor authentication into this website would have been to use some of the industry standard platforms available to the

public. Authentication platforms such as Amazon Cognito, Twilio Authy, Auth0, and KeyCloak are some ways that websites can authenticate their clients with other forms of authentication. These other authentication factors include the following: SMS, push notifications, email, and voice authentication if you have a specified app installed on your device.