

Database driven website application - 2115745

by Laurice Keogh

Submission date: 05-Dec-2022 10:36AM (UTC+0000)

Submission ID: 191030575

File name: 35959_Laurice_Keogh_Database_driven_website_application_-
_2115745_460452_832101840.docx (1.9M)

Word count: 1207

Character count: 6329

Name: Laurice Keogh

Student ID: 2115745

Word count: 1048

Database driven website application

Table of Contents

Introduction	1
Part A – Overview of two security issues.....	1
Part B – Implementation of defences against security issues	2
SQL injection prevention in my code	2
Broken authentication prevention for my website	5
References	8

Introduction

There are many security issues to consider when producing a website. Hackers are able to gain access to all of the information stored in the linked database if precautions aren't taken by the producer of the website. This poses a risk to people's personal information as hackers can make your information publicly available or sell it on.

Part A – Overview of two security issues

For a website that's linked to a database using SQL the programmer must be aware of hackers trying to access the database records by using SQL injections. [1] SQL injection is where a hacker inputs a malicious SQL statement into the website. [1] Hackers can also write injections which can add, edit, or delete records in the database. There are four main ways that hackers can use SQL Injection. [2] Error based SQL injection can be done by writing a query to return errors into the web address, giving insight into the structure of the database. [3] This is a type of in-band SQL injection. [2] This can be avoided by disabling error messages before launching the website. Another type is Boolean based which is used when the producer of the website has turned the error display off. Hackers instead look for vulnerabilities in the page by inserting a condition into the query (through the web address). They will insert a false statement and if the page doesn't work as normal or there isn't a result then it can be assumed there is a vulnerability. Hackers can usually work out the version of software and gain access to the database. [2] This is classed as a blind SQL injection. Another blind SQL injection is a time-based SQL injection. A time-based injection works by the hacker telling the page to sleep which if there is a vulnerability the page response time will slow down. Extraction is taken even though there's no changes on the screen. The fourth type of SQL injection is out-of-band which means data has to be sent from the database server to the attacker's device. [2] Depending on the expertise of the hacker they might be able to take over the entire database and website application. Ways of preventing SQL injection include using prepared statements

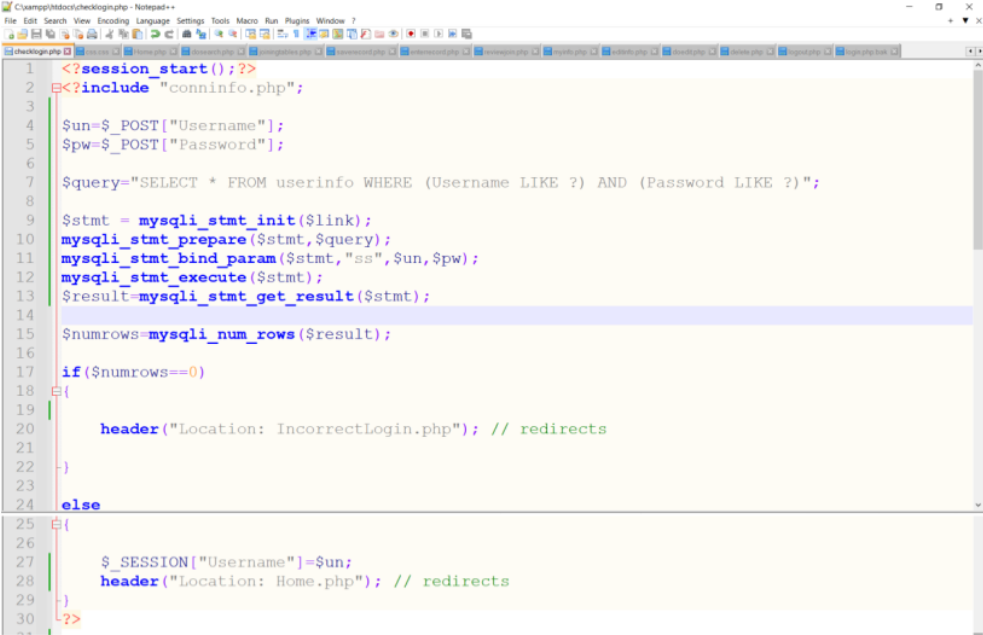
Another security issue to be aware of is broken authentication [4] which is where the hacker impersonates an authorised user. The identity can be stolen through the session or the credential management. [4] A stolen session id is just as useful as log in details. The main way this takes place is someone logging in on a device and not logging out, then the device gets used by someone else who

impersonates them. This could be avoided by having a time out period on sessions. Sometimes the session ID is displayed in the URL which someone else on the same network could see and use to gain access to the database records. This can be avoided by excluding the session ID from the website URL. [4] Session fixation attacks can occur when the session key always stays the same. An attacker might send a link which includes a predetermined session key and if the user logs in then the attacker can hijack the session. This attack could be avoided if a legitimate user is given a new session ID after logging in. [5] Storing passwords as clear text leaves a vulnerability because if a user uses the same password on multiple different places, they are at risk of their data being hacked elsewhere or having their passwords/data sold on. This can be avoided by hashing passwords. Hackers often use credential stuffing to gain access, so it is important for a website to add extra password requirements for added security. [5] Some of the impacts of broken authentication include data breaches, impersonation, personal data being exposed and hackers gaining administrator access to make changes.

Part B – Implementation of defences against security issues

SQL injection prevention in my code

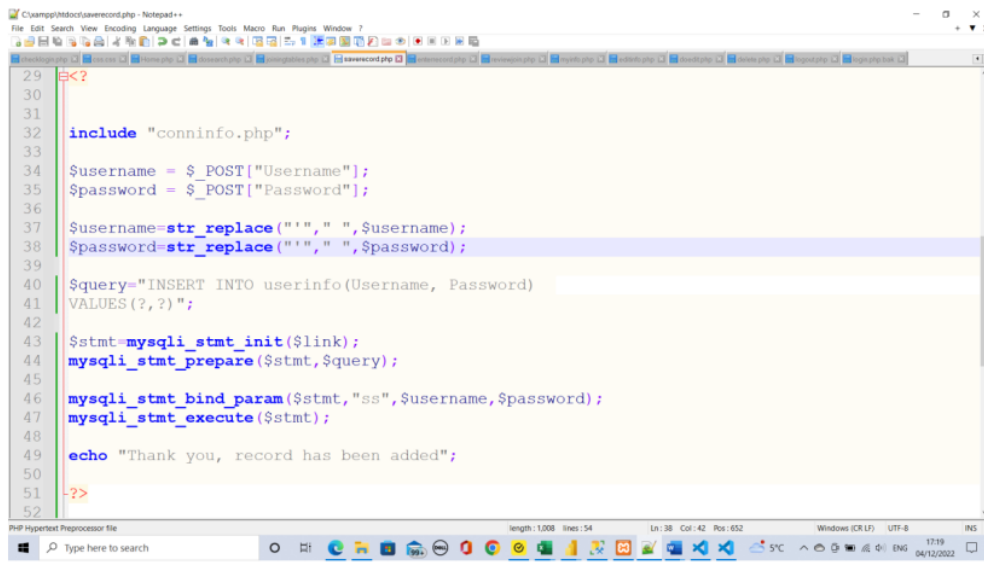
checklogin.php



```
1 <?session_start();?>
2 <?include "conninfo.php";
3
4 $un=$_POST["Username"];
5 $pw=$_POST["Password"];
6
7 $query="SELECT * FROM userinfo WHERE (Username LIKE ?) AND (Password LIKE ?)";
8
9 $stmt = mysqli_stmt_init($link);
10 mysqli_stmt_prepare($stmt,$query);
11 mysqli_stmt_bind_param($stmt,"ss",$un,$pw);
12 mysqli_stmt_execute($stmt);
13 $result=mysqli_stmt_get_result($stmt);
14
15 $numrows=mysqli_num_rows($result);
16
17 if($numrows==0)
18 {
19     header("Location: IncorrectLogin.php"); // redirects
20 }
21
22 }
23
24 else
25 {
26     $_SESSION["Username"]=$un;
27     header("Location: Home.php"); // redirects
28 }
29
30 <?>
```

This prepared statement ensures that SQL cannot be used to edit the query, therefore protecting all of the data in the database by refusing entry unless a genuine username and password is used. If a hacker tried to inject an SQL statement they would be redirected to the incorrect login in page.

Saverecord.php



```
29 <?
30
31
32 include "conninfo.php";
33
34 $username = $_POST["Username"];
35 $password = $_POST["Password"];
36
37 $username=str_replace("'", " ", $username);
38 $password=str_replace("'", " ", $password);
39
40 $query="INSERT INTO userinfo (Username, Password)
41 VALUES (?, ?) ";
42
43 $stmt=mysqli_stmt_init($link);
44 mysqli_stmt_prepare($stmt,$query);
45
46 mysqli_stmt_bind_param($stmt,"ss", $username,$password);
47 mysqli_stmt_execute($stmt);
48
49 echo "Thank you, record has been added";
50
51 -?>
52
```

This prepared statement ensures that SQL statements can't be inserted and stored in the database which would be a major threat. It also ensures that any apostrophise entered do not interfere with the SQL because they are automatically replaced with a space.

[editinfo.php](#)

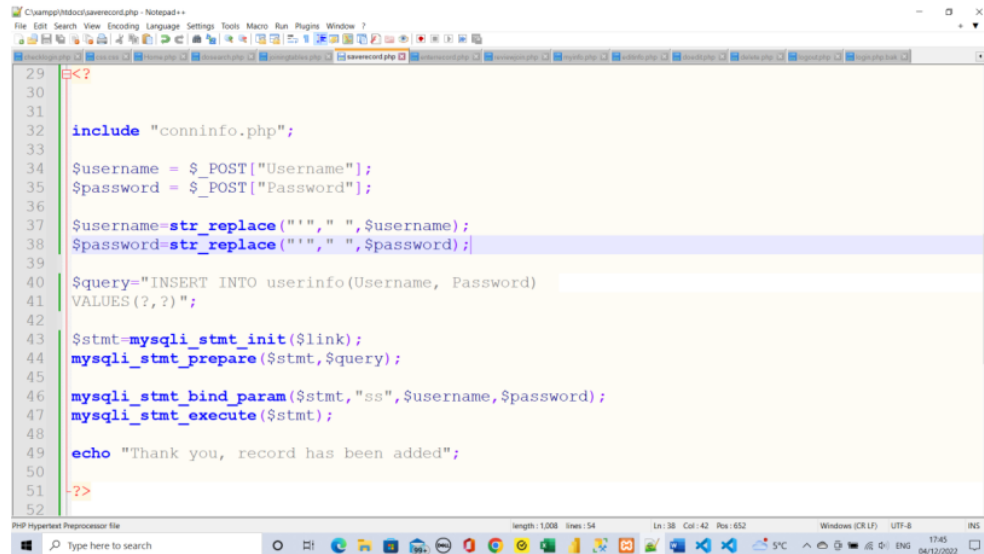
```
C:\xampp\htdocs\editinfo.php - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
PHP Hypertext Preprocessor file length: 1,552 lines: 63 Ln: 62 Col: 5 Pos: 1,549 Windows (CR LF) UTF-8 INS
1 <?
2 session_start();
3 >?
4 <style><?php include 'C:/xampp/htdocs/css.css'; ?></style>
5 <style>.human{display:none; visibility:hidden}</style>
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <meta charset="utf-8" />
10 <title> Book Catalogue </title>
11 <link rel="stylesheet" type="text/css" href="css.css">
12 </head>
13
14 <body>
15 <header id="main-header">
16 <div class="container">
17 <h1> Book Catalogue </h1>
18 </div>
19 </header>
20
21 <nav id="navbar">
22 <div class="container">
23 <ul>
38 <?
39 include "conninfo.php";
40
41 $username = $_POST["username"];
42 // $username = 'lauricek';
43 // $username = $_GET["username"];
44 $query="SELECT * FROM userinfo WHERE Username LIKE '$username'";
45 $result=mysqli_query($link,$query);
46 // $r = mysqli_fetch_array($result); // one row
47 $r = mysqli_fetch_array($result);
48
49 >?
50
51 <form action="doedit.php" method="post">
52
53 Password: <input type="text" name="Password" value="<?echo $r["Password"];?>"><br>
54 Postcode: <input type="text" name="Postcode" value="<?echo $r["Postcode"];?>"><br>
55 Mobile Number: <input type="text" name="MobileNumber" value="<?echo $r["MobileNumber"];?>"><br>
56 <input type="text" name="extra" class="human">
57 </br>
58 <input type="hidden" name="Username" value="<?echo $username;?>">
59 <input type="submit" value="Edit">
60
```

There is a hidden data entry field that isn't visible to humans but it would be visible to a bot looking at the code. This prevents a bot system from altering or corrupting the database through injection as it doesn't allow their changes to be made while the field isn't empty.

[joiningtables.php](#)

There is no encryption here for the user's password details which leaves them vulnerable and displayed in plain text. Hash encryption would be beneficial here.

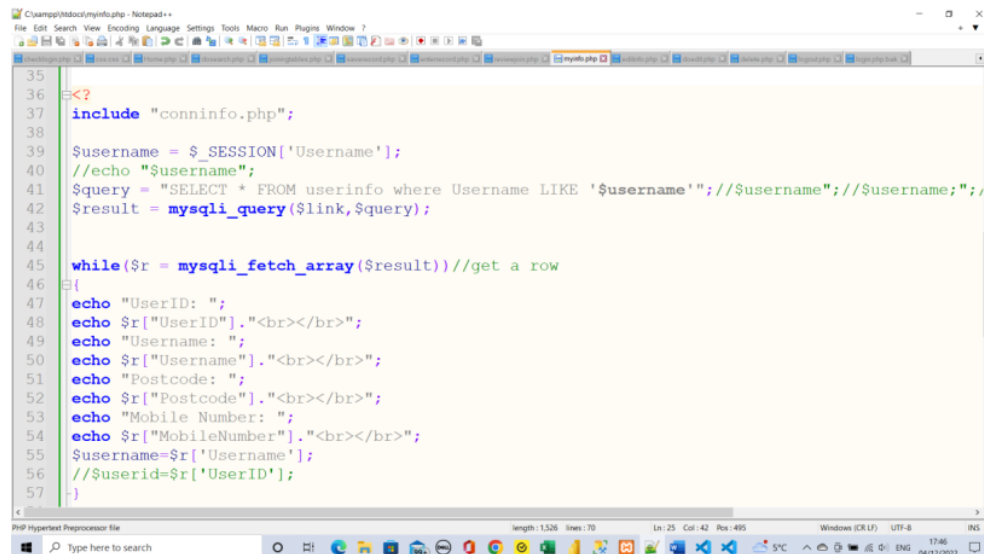
saverecord.php



```
29 <?
30
31
32 include "conninfo.php";
33
34 $username = $_POST["Username"];
35 $password = $_POST["Password"];
36
37 $username=str_replace(" ", "", $username);
38 $password=str_replace(" ", "", $password);
39
40 $query="INSERT INTO userinfo(Username, Password)
41 VALUES(?, ?) ";
42
43 $stmt=mysqli_stmt_init($link);
44 mysqli_stmt_prepare($stmt,$query);
45
46 mysqli_stmt_bind_param($stmt,"ss", $username,$password);
47 mysqli_stmt_execute($stmt);
48
49 echo "Thank you, record has been added";
50
51 -?>
```

There is no encryption here for the user's password details which leaves them vulnerable and displayed in plain text. Hash encryption would be beneficial here.

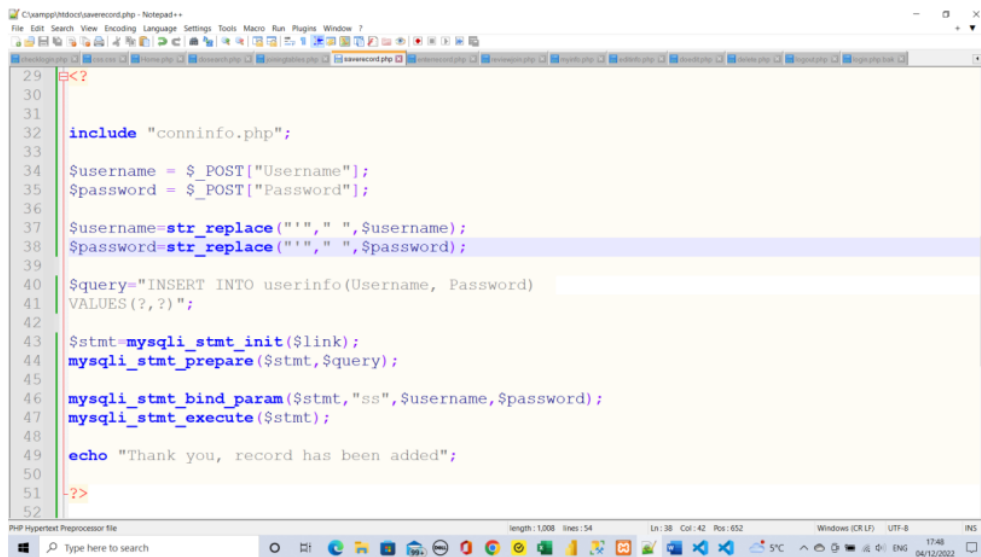
myinfo.php



```
35
36 <?
37 include "conninfo.php";
38
39 $username = $_SESSION['Username'];
40 //echo "$username";
41 $query = "SELECT * FROM userinfo where Username LIKE '$username'";//$username;//$username";
42 $result = mysqli_query($link,$query);
43
44
45 while($r = mysqli_fetch_array($result)){//get a row
46 {
47 echo "UserID: ";
48 echo $r["UserID"]."<br></br>";
49 echo "Username: ";
50 echo $r["Username"]."<br></br>";
51 echo "Postcode: ";
52 echo $r["Postcode"]."<br></br>";
53 echo "Mobile Number: ";
54 echo $r["MobileNumber"]."<br></br>";
55 $username=$r['Username'];
56 // $userid=$r['UserID'];
57 }
```

The user's info including their password is displayed as clear text which isn't secure so the security would be helped by displaying this information in the hash encryption format.

enterrecord.php



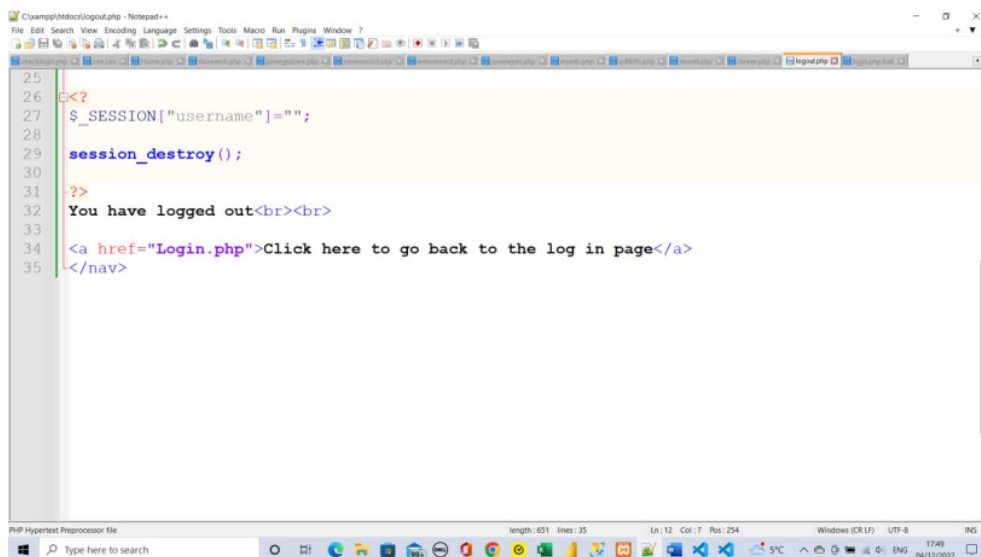
```
29 <?
30
31
32 include "conninfo.php";
33
34 $username = $_POST["Username"];
35 $password = $_POST["Password"];
36
37 $username=str_replace("'", "", $username);
38 $password=str_replace("'", "", $password);
39
40 $query="INSERT INTO userinfo(Username, Password)
41 VALUES (?, ?) ";
42
43 $stmt=mysqli_stmt_init($link);
44 mysqli_stmt_prepare($stmt,$query);
45
46 mysqli_stmt_bind_param($stmt,"ss", $username, $password);
47 mysqli_stmt_execute($stmt);
48
49 echo "Thank you, record has been added";
50
51 -?>
52
```

The user can enter any password they like which means they are free to enter common passwords and leave themselves vulnerable to credential stuffing attacks. My site would benefit from having some input length and character requirements before their account can be added to the database.

All pages

The session ID is hidden on all of my webpage URLs which adds a layer of security because it prevents a hacker gaining access through using the ID as it is just as useful to gain access to a hacker as log in credentials are.

The session never gets destroyed unless they manually log out using logout.php (shown below) and this means that if someone logs in on a public device and doesn't log out then someone is able to impersonate them. My website would benefit from a session destroy after a certain period of time to minimise the chance of this happening.



```
25
26
27 <?
28 $_SESSION["username"]="";
29 session_destroy();
30
31 ?>
32 You have logged out<br><br>
33
34 <a href="Login.php">Click here to go back to the log in page</a>
35 </nav>
```

References

[1]

C. Labs, "SQL Injection Vulnerability: What is it and how to stay protected? | Cyware Hacker News," *Cyware Labs*. <https://cyware.com/news/sql-injection-vulnerability-what-is-it-and-how-to-stay-protected-669cd8db/>

[2]

"What is the SQL Injection Vulnerability & How to Prevent it?," *www.invicti.com*. <https://www.invicti.com/blog/web-security/sql-injection-vulnerability/#TypesofSQL>

[3]

"In-Band SQL Injection | Learn AppSec," *Invicti*, Mar. 28, 2022. <https://www.invicti.com/learn/in-band-sql-injection/>

[4]

D. Poza, "What Is Broken Authentication?," *Auth0 - Blog*, Aug. 20, 2020. <https://auth0.com/blog/what-is-broken-authentication/>

[5]

"Broken Authentication Vulnerability," *GeeksforGeeks*, Dec. 30, 2019. <https://www.geeksforgeeks.org/broken-authentication-vulnerability/>

Database driven website application - 2115745

GRADEMARK REPORT

FINAL GRADE

68/100

GENERAL COMMENTS

Instructor

WEBSITE

Database structure and design - OK - a little bit limited in information about each book - could have included author etc. However, suitably normalised. 9/10

Range of PHP demonstrated -

Search is OK - would be nice to search by genre too, to allow the user to discover books via your website. The search relies on you knowing the title of the book you're looking for (although the wild card is there)

Be careful with Register - this needs validation. I was able to create a blank account and then use it to log in.

Edit is OK, but it would help the user if you populate the fields with the current information. Like register, I was able to edit it to blank fields, so this also need validation.

Delete is OK, although a bit "sudden" - you might want to prompt the user eg "Are you sure?"

30/50

Overall integration of website -

Layout is a little sparse, particularly on the home page.
Never make the user log in before they can do anything - they will quickly lose interest in your site.
Have at least some information that is publicly available.

Some pages are not styled.

5/10

REPORT

Investigation/analysis of two security issues -

SQL Injection Generally good and evidence of research undertaken. Add some code examples in to illustrate your points - this will support the words and make it clearer to the reader.

Broken authentication is good - again, need some technical examples 11 /15

Mitigation of security issues in your site -

Prepared statements used appropriately. You have identified suitable Broken Authentication areas to address well. 13/15

General Comments: Don't forget to put a short Conclusion in - all academic reports should have this (along with an Introduction)

TOTAL: 68/100

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8
