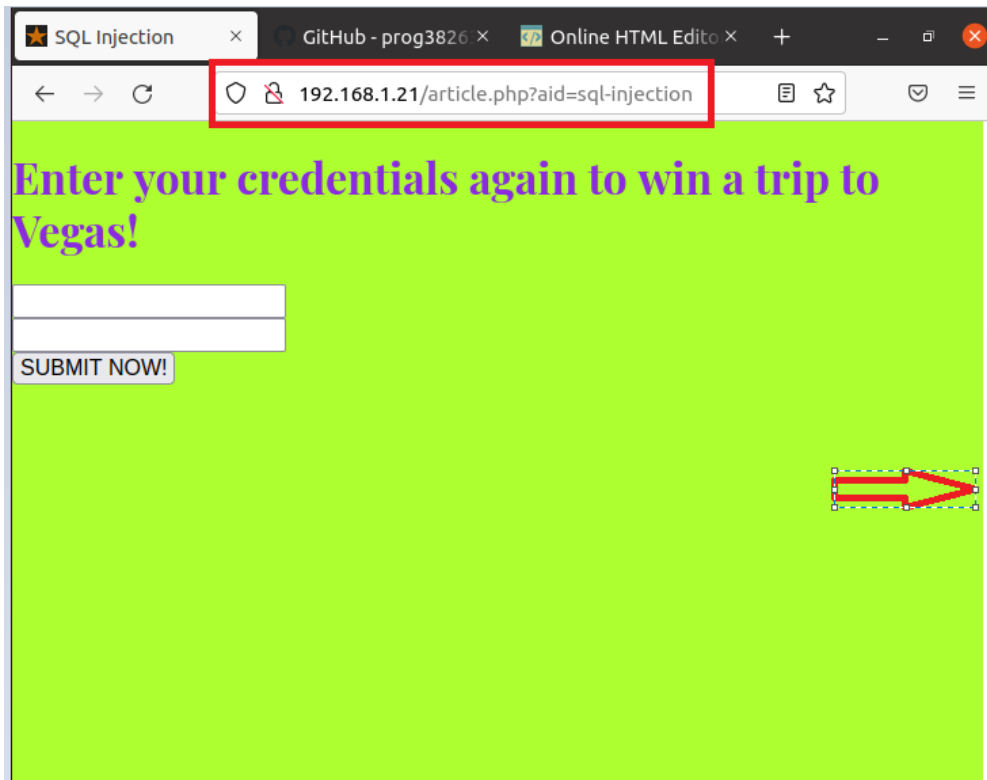


Member #1: Jackson Yuan
Member #2: Cristian Di Bartolomeo

V-1: Cross-Site Scripting (XSS)

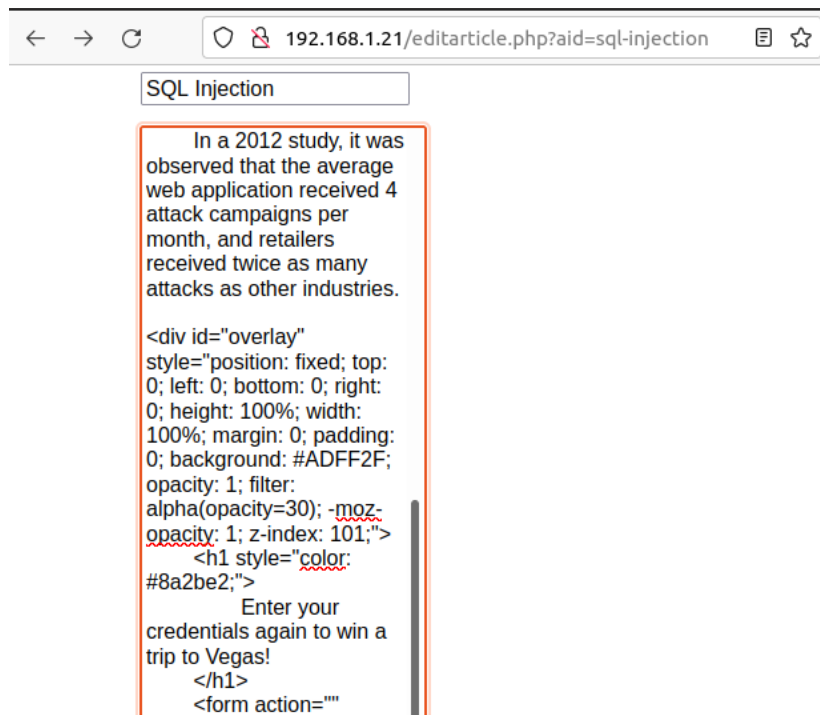


(Screenshot above for before the user clicks the “read more...” link)



(Screenshot after the user clicks the “read more...” link; notice the scroll bar is in the middle thus locking the user to only this screen)

Explanation: For xss the exploit here was pretty simple, during the edition phase for each article, one can write HTML or JavaScript into the editing textbox thus falling into the trap of attackers where they can manipulate blog. The severity of this exploit is that it forces the user into thinking that potentially the website itself is showing these images but in fact it’s that of the code of the attacker injected and executed by the user’s browser.

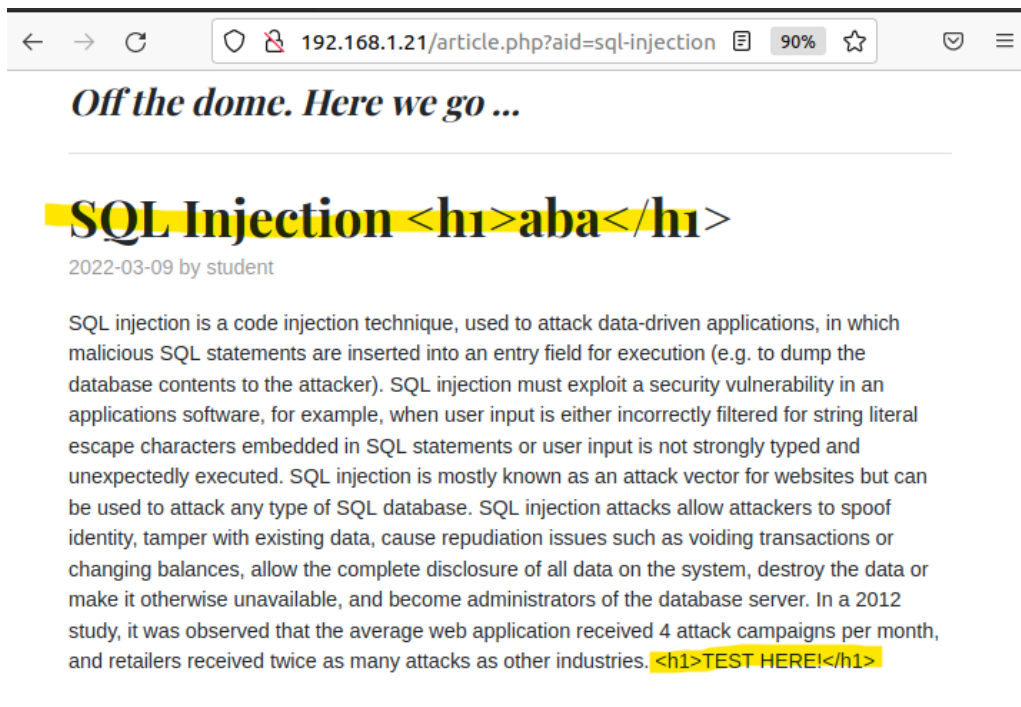


(Code of the xss; here you can see it was written in HTML right in the blog text field!)

Prevention and Mitigation: The way we prevent such attack is to simply apply proper validation and sanitation to the data. This can be using escaping HTML tag, HTML attribute, CSS value, URL value, etc. We can also not allow HTML/JavaScript special characters from fields as well as Strip HTML/JavaScript special characters (e.g., using `htmlspecialchars()` function) and treat them all as a string text.

Security Fix: The security fix was to encode the `htmlspecialchars()` function into the output area.

```
editarticle.php  x    reflected.php  x    article.php  x
15 <head>
16 <title><?php echo htmlspecialchars($row['title'], ENT_QUOTES, 'UTF-8'); ?></
   title>
17     <?php include("templates/header.php"); ?>
18
19
20
21 </head>
22 <body>
23     <?php include("templates/nav.php"); ?>
24     <?php include("templates/contentstart.php"); ?>
25
26     <h3 class="pb-4 mb-4 font-italic border-bottom">
27         Off the dome. Here we go ...
28     </h3>
29
30     <div class="blog-post">
31         <h2 class="blog-post-title"><?php echo
           htmlspecialchars($row['title'], ENT_QUOTES, 'UTF-8'); ?></h2>
32         <p class="blog-post-meta">
33             <?php echo substr($row['date'], 0, 10). " by ".
               $row['author'] ?>
34         </p><p>
35             <?php echo htmlspecialchars($row['content'], ENT_QUOTES,
               'UTF-8') ?>
```



(Screenshot above shows after running the htmlspecialchars() function)

V-2: SQL Injection

Please sign in

Sign in



File Actions Edit View Help

```
(x03@talos) - [~]  
$ Cristian Di Bartolomeo 991521498
```

Back to top

[Blog](#) [Home](#) [Admin](#)

[Logout admin](#)

Article Management

[New Post +](#)

Post Title	Author	Date	Modify	Delete
test3	student	2022-03-13		
test2	admin	2022-03-13		
test1	admin	2022-03-13		
SQL Injection	student	2022-02-16		
Cross-site Scripting	admin	2022-02-16		



File Actions Edit View Help

```
(x03@talos) - [~]  
$ Cristian Di Bartolomeo 991521498
```

Explanation: Poor/lack of user input sanitization allows a malicious actor to implement escape characters to exit a string and use dangerous SQL statements. By modifying the SQL query, an attacker can go around SQL's checks and gain access to things they shouldn't be allowed to, like accounts.

Prevention and Mitigation: Fixing SQL injections can almost be done entirely by implementing input sanitization. Input sanitization would watch for dangerous characters being used and effectively pass them through without compromising database security.

V-3: Broken Access Control

```
x03@talos: ~  
File Actions Edit View Help  
  
(x03@talos) ~  
$ curl 'http://10.0.69.79/article.php?aid=test1' -X "GET" | head -n 4  
% Total % Received % Xferd Average Speed Time Time Time Current  
Dload Upload Total Spent Left Speed  
100 2691 0 2691 0 0 875k 0 --:--:-- --:--:-- --:--:-- 875k  
<!doctype html>  
<html lang="en">  
<head>  
<title>test1</title>  
  
(x03@talos) ~  
$ curl 'http://10.0.69.79/deletearticle.php?aid=test1' -X "GET"  
  
(x03@talos) ~  
$ curl 'http://10.0.69.79/article.php?aid=test1' -X "GET" | head -n 4  
% Total % Received % Xferd Average Speed Time Time Time Current  
Dload Upload Total Spent Left Speed  
100 2826 0 2826 0 0 1379k 0 --:--:-- --:--:-- --:--:-- 1379k  
<br />  
<b>Warning</b>: pg_fetch_array(): Unable to jump to row 0 on PostgreSQL result index 140631479427168 in <b>/code/arti  
cle.php</b> on line <b>11</b><br />  
<!doctype html>  
<html lang="en">  
  
(x03@talos) ~  
$ Cristian Di Bartolomeo 991521490
```

Explanation: As the deletearticle.php script does not have any checks before executing, any requests to a specific article will cause the website to delete it without checking if the user should even be allowed to do so. In the image above, Curl can be used to delete the post “test 1” despite the fact that we never established any sessions, meaning the action was performed unauthenticated.

Prevention and Mitigation: To prevent this, it’s important to make sure there are access controls on any scripts like deletearticle.php. In the example below, this is fixed by checking if the user is admin or if they are the author of the post.

```
deletearticle.php - Mousepad  
File Edit Search View Document Help  
  
Warning, you are using the root account, you may harm your system.  
  
<?php  
session_start();  
include("config.php");  
include("lib/db.php");  
  
$aid = $_GET['aid'];  
$result = get_article_list($dbconn);  
while ($row = pg_fetch_array($result)) {  
    if ($_SESSION['username'] == 'admin' or $_SESSION['username'] == $row['author']) {  
        #echo "aid=".$aid."<br>";  
        $result = delete_article($dbconn, $aid);  
        #echo "result=".$result."<br>";  
        # Check result  
        header("Location: /admin.php");  
    }  
}  
  
?>
```



```
x03@talos: ~  
File Actions Edit View Help  
  
(x03@talos) - [~]  
$ curl 'http://10.0.69.79/article.php?aid=a' -X "GET" | head -n 4  
% Total % Received % Xferd Average Speed Time Time Time Current  
Dload Upload Total Spent Left Speed  
100 2683 0 2683 0 0 1310k 0 --:--:-- --:--:-- --:--:-- 1310k  
<!doctype html>  
<html lang="en">  
<head>  
<title>a</title>  
  
(x03@talos) - [~]  
$ curl 'http://10.0.69.79/deletearticle.php?aid=a' -X "GET"  
  
(x03@talos) - [~]  
$ curl 'http://10.0.69.79/article.php?aid=a' -X "GET" | head -n 4  
% Total % Received % Xferd Average Speed Time Time Time Current  
Dload Upload Total Spent Left Speed  
100 2683 0 2683 0 0 873k 0 --:--:-- --:--:-- --:--:-- 1310k  
<!doctype html>  
<html lang="en">  
<head>  
<title>a</title>  
  
(x03@talos) - [~]  
$
```

(Above - Unable to delete the page without authentication)

V-4: Missing Role-Based Access Control

Logout Student

Article Management

New Post +

Post Title	Author	Date	Modify	Delete
test3	student	2022-03-13		
test2	admin	2022-03-13		
test1	admin	2022-03-13		
SQL Injection	student	2022-02-16		
Cross-site Scripting	admin	2022-02-16		

x03@talos: ~
File Actions Edit View Help

(x03@talos) - [~]
\$ Cristian D1 Bartolomeo 991521498

Explanation: The student user is able to modify and delete the admin's pages when they shouldn't be able to. This is because the student user is just listed as "user". which should be less privileged than the admin user.

Prevention and Mitigation: To prevent this, the PHP page should only display pages that belong the user if they're not an admin. In the screenshots below, I implement the code that checks if the user is admin. If the user is admin, they can see and modify all pages. If the user is not an admin, it'll only display the user's own pages.

admin.php - Mousepad

File Edit Search View Document Help

Warning, you are using the root account, you may harm your system.

```
<title>Admin</title>
<?php include("templates/header.php"); ?>
</head>
<body>
  <?php include("templates/nav.php"); ?>
  <?php include("templates/contentstart.php"); ?>

<h2>Article Management</h2>

<p><button type="button" class="btn btn-primary" aria-label="Left Align" onclick="window.location='/newarticle.php';">
New Post <span class="fa fa-plus" aria-hidden="true"></span>
</button></p>

<table class="table">
<tr><th>Post Title</th><th>Author</th><th>Date</th><th>Modify</th><th>Delete</th></tr>

<?php
# get articles by user or, if role is admin, all articles
if ($SESSION['username'] == 'admin'){
  $result = get_article_list($dbconn);
  while ($row = pg_fetch_array($result)) {
    ?>
<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>
  <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>
</tr>
<?php } } //close while loop ?>

<?php
# get articles by user or, if role is admin, all articles
if ($SESSION['username'] != 'admin'){
  $result = get_article_list($dbconn);
  while ($row = pg_fetch_array($result)) {
    if ($SESSION['username'] == $row['author']){
      ?>
<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>
  <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>
</tr>
<?php } } } //close while loop ?>
</table>
<?php include("templates/contentstop.php"); ?>
<?php include("templates/footer.php"); ?>
</body>
</html>
```

cristian@dibart-Debian22: /mnt/hgfs

File Edit Tabs Help

cristian@dibart-Debian22: /mnt/hgfs\$ Cristian Di Bartolomeo 991521490

Blog Home Admin Logout student

Article Management

New Post +

Post Title	Author	Date	Modify	Delete
test3	student	2022-03-13		
SQL Injection	student	2022-02-16		

x03@talos: ~

File Actions Edit View Help

~(x03@talos) - [~]

\$ Cristian Di Bartolomeo 991521490

Screenshot above shows that the student user is only able to modify their own pages.

V-5: Insecure password handling and storage

Select: authors

Select data

Show structure

Alter table

New item

Select

Search

Sort

Limit50

Text length100

ActionSelect

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

<input type="checkbox"/> Modify	id	created_on	username	password	role
<input type="checkbox"/> edit	1	2022-02-16 18:30:27.218837+00	admin	password123	admin
<input type="checkbox"/> edit	2	2022-02-16 18:30:27.220178+00	student	password123	user

Whole result

☐ 2 rows

Modify

Save

Selected (0)

EditCloneDelete

Export (2)

Import

FileActionsEditViewHelp

(x03 talos) - [~]
\$ Cristian Di Bartolomeo 991521498

(Screenshot above shows an example of insecure password handling and storage)

Explanation: The severity and impact of insecure password handling and storage is extremely high as any malicious access/reads to the database would mean all accounts would be susceptible to compromise from malicious actors.

Prevention and Mitigation: To fix this issue, we would either store it with strict permissions in which the logs are not publicly exposed and the logs should be on a separate partition. Finally, these passwords should be encrypted with non-broken crypto algorithms such as SHA256 or bcrypt and run those through multiple iterations, salt then hash it afterwards.

Security Fix: The security fix would be to encrypt using the supported algorithms on postgres DB; in this case the crypt() function supports bf which is the best option as the others are subpar.

Table F-18. Supported algorithms for `crypt()`

Algorithm	Max password length	Adaptive?	Salt bits	Description
bf	72	yes	128	Blowfish-based, variant 2a
md5	unlimited	no	48	MD5-based crypt
xdes	8	yes	24	Extended DES
des	8	no	12	Original UNIX crypt

(Screenshot above shows the available algorithms for the `crypt()` function)

PostgreSQL » db » postgres » public » SQL command

SQL command

CREATE EXTENSION IF NOT EXISTS pgcrypto

Query executed OK, 0 rows affected. (0.000 s) [Edit](#)

CREATE EXTENSION IF NOT EXISTS pgcrypto;

(Screenshot above creates the extension `pgcrypto`)

← → ↻ 192.168.1.21:8080/?pgsql=db&username=postgres&c ☆

PostgreSQL » db » postgres » public » SQL command Logout

SQL command Jackson Yuan
991302821

```
INSERT INTO authors (username, password) VALUES ('UpdatedAdmin', crypt('password123', gen_salt('bf')))
```

Query executed OK, 1 row affected. (0.006 s) [Edit](#)

```
INSERT INTO authors (username, password) VALUES ('UpdatedAdmin', crypt('password123', gen_salt('bf')))
```

(Screenshot above shows we create a new user in which we salt and encrypt the given password when calling the crypt() function; this case we will use the blowfish algorithm as it's the strongest supported one)

SQL command Jackson Yuan
991302821

```
INSERT INTO authors (username, password) VALUES ('UpdatedStudent', crypt('password123', gen_salt('bf')))
```

Query executed OK, 1 row affected. (0.007 s) [Edit](#)

(Screenshot above now applying the same encryption algorithm as aforementioned above to the second user student)









created_on	username	password
2022-03-09 02:32:18.144365+00	admin	password123
2022-03-09 02:32:18.14574+00	student	password123
2022-03-16 04:37:23.832065+00	UpdatedStudent	\$2a\$06\$MDZWfJi8BZ4VNPFR6hEou8Wm97HaHQXs27PKZp1aT8Cacb1iJ/R.
2022-03-16 04:41:49.572918+00	UpdatedAdmin	\$2a\$06\$30LVaDv41TGf2IgRfygQye6UEslo/Bb3KiqqDSELHnDMk.W5OHGu

(Screenshot above shows the highlighted updated passwords for both admin and student)

V-6: CSRF

Article Management

New Post +

Post Title	Author	Date	Modify	Delete
bob	student	2022-03-14		
cheeky change	student	2022-03-14		
SQL Injection	student	2022-03-09		
aba				
Cross-site Scripting	admin	2022-03-09		

(Screenshot above shows two articles created, cheek change will be the CSRF exploit)

Since XSS is shown to work in the previous exploit, we embed HTML code in which we redirect to the article called "deletearticle.php" in which we will delete the article called "bob". See screenshot below:

← → ↻

🛡️ 192.168.1.21/editarticle.php?aid=cheeky-change ☆

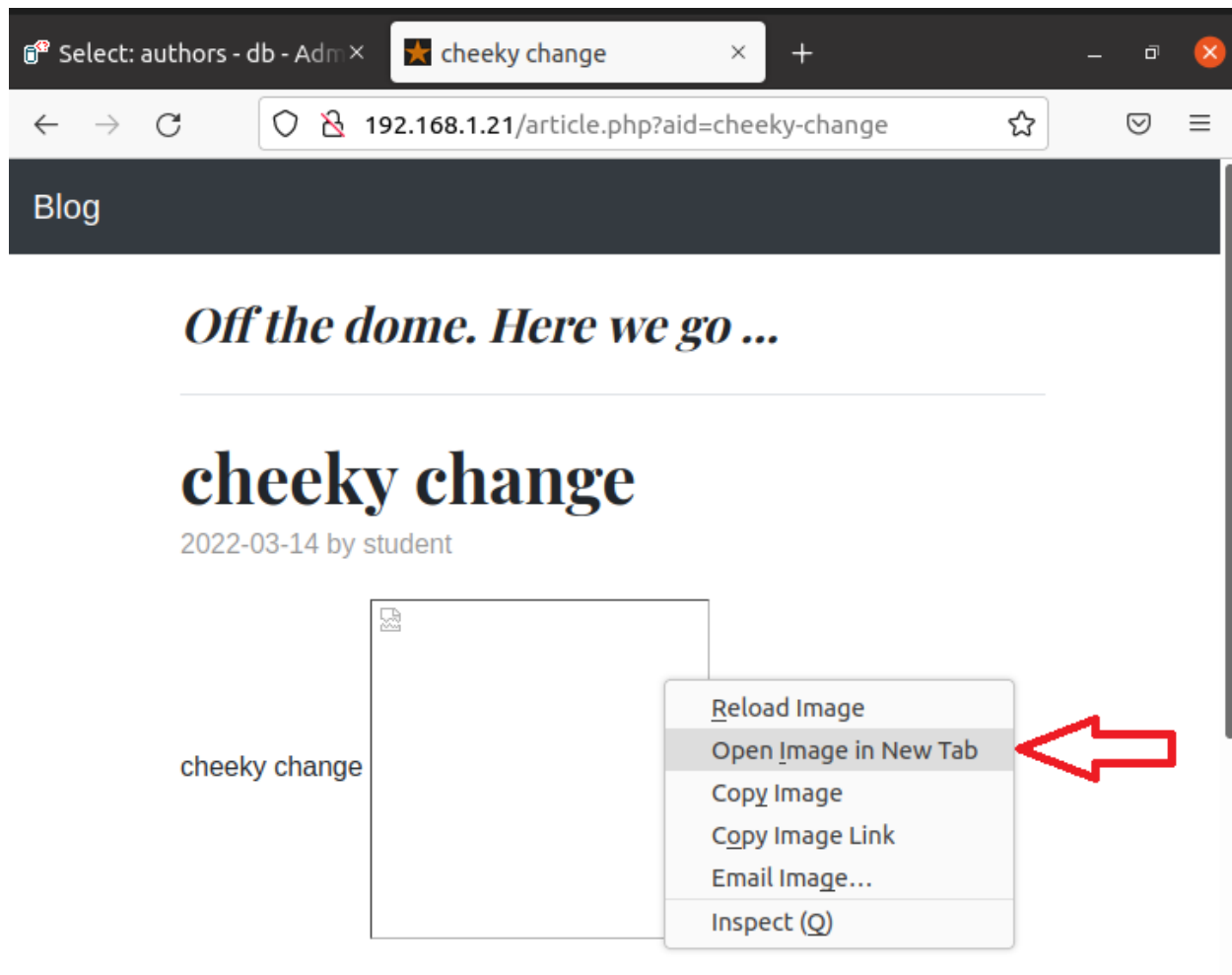
New Post

cheeky change

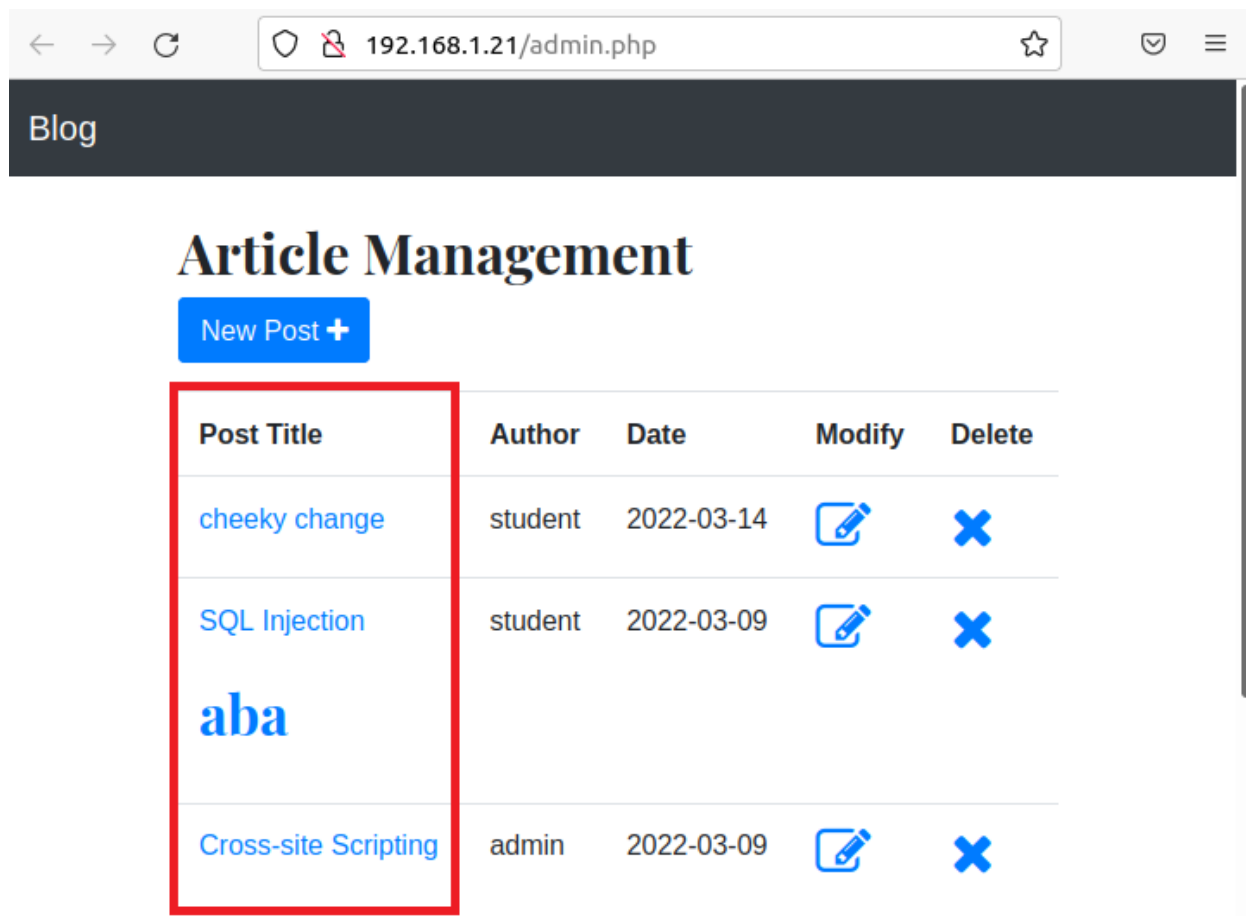
cheeky change

```
<html>  
  
  <body>  
  
      
  
  </body>  
  
</html>
```

Update



(Screenshot above shows the image link to the malicious code when opening in a new tab)



(Screenshot above shows that the article “bob” is deleted once user clicks on the image link to the malicious code as aforementioned above)

Explanation: The severity and impact of this vulnerability is that it allows the attacker to submit their own request to an authenticated application without the user’s acknowledgement. As you can see, students or anyone can delete articles by a simple XSS injection exploit as aforementioned.

Prevention and Mitigation: One simple way of doing this is by removing escape characters from being entered into the text field as aforementioned in V-1. This is done by using the htmlspecialchars() functions.

Security Fix: The security fix was to encode the htmlspecialchars() function into the text field area.

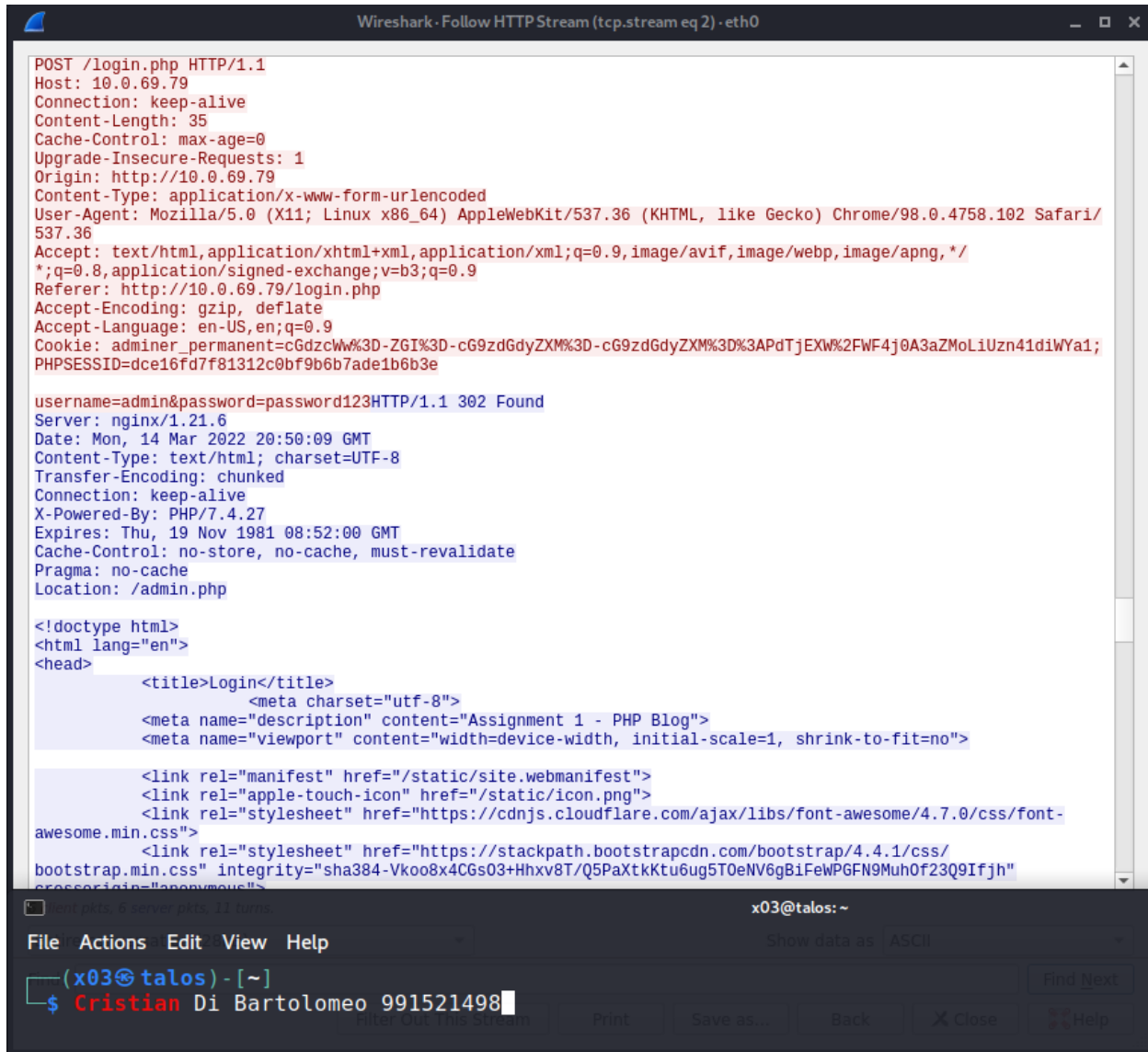
The image shows a code editor with three tabs: 'editarticle.php', 'reflected.php', and 'article.php'. The 'article.php' tab is active, displaying PHP code for an article page. The code includes a title, a header, a body with a navigation bar and content start, and a main content area. The title and content are rendered using `htmlspecialchars()` to prevent XSS attacks. The code is as follows:

```
15 <head>
16 <title><?php echo htmlspecialchars($row['title'], ENT_QUOTES, 'UTF-8'); ?></title>
17 <?php include("templates/header.php"); ?>
18
19
20
21 </head>
22 <body>
23 <?php include("templates/nav.php"); ?>
24 <?php include("templates/contentstart.php"); ?>
25
26 <h3 class="pb-4 mb-4 font-italic border-bottom">
27 Off the dome. Here we go ...
28 </h3>
29
30 <div class="blog-post">
31 <h2 class="blog-post-title"><?php echo
32 htmlspecialchars($row['title'], ENT_QUOTES, 'UTF-8'); ?></h2>
33 <p class="blog-post-meta">
34 <?php echo substr($row['date'], 0, 10). " by ".
35 $row['author'] ?>
36 </p><p>
37 <?php echo htmlspecialchars($row['content'], ENT_QUOTES,
38 'UTF-8') ?>
39 </p></div>
40
```

Below the code editor is a screenshot of a web browser. The address bar shows the URL `192.168.1.21/article.php?aid=cheeky-change`. The browser displays a blog post titled "cheeky change" by student, dated 2022-03-14. The post content is "cheeky change" followed by an HTML payload: `<html> <body> </body> </html>`. The browser also shows a footer with the text "This blog and contents © 2022" and a "Back to top" link.

(Screenshot above shows escape characters being printed out as a string)

V-7: The entire website, including the login page, is served over plaintext HTTP



```
Wireshark - Follow HTTP Stream (tcp.stream eq 2) - eth0

POST /login.php HTTP/1.1
Host: 10.0.69.79
Connection: keep-alive
Content-Length: 35
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://10.0.69.79
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.102 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://10.0.69.79/login.php
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Cookie: adminer_permanent=cGdzcWw%3D-ZGI%3D-cG9zdGdyZXN%3D-cG9zdGdyZXN%3D%3APdTjEXW%2FWF4j0A3aZMoLiUzn41diWYa1; PHPSESSID=dce16fd7f81312c0bf9b6b7ade1b6b3e

username=admin&password=password123HTTP/1.1 302 Found
Server: nginx/1.21.6
Date: Mon, 14 Mar 2022 20:50:09 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
X-Powered-By: PHP/7.4.27
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: /admin.php

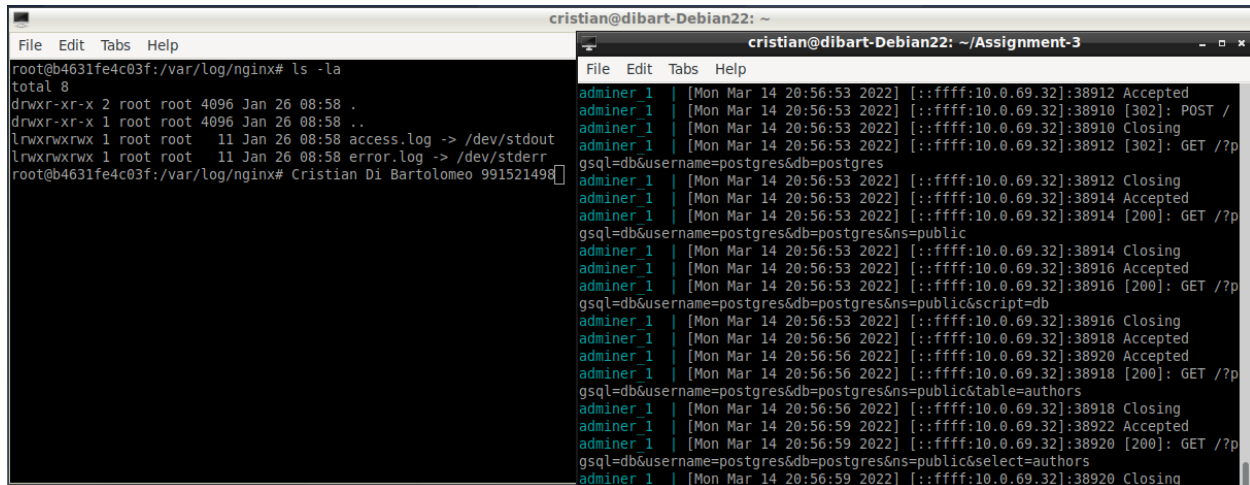
<!doctype html>
<html lang="en">
<head>
  <title>Login</title>
  <meta charset="utf-8">
  <meta name="description" content="Assignment 1 - PHP Blog">
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
  <link rel="manifest" href="/static/site.webmanifest">
  <link rel="apple-touch-icon" href="/static/icon.png">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/font-awesome/4.7.0/css/font-awesome.min.css">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGs03+Hhvx8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">
</head>
```

(Screenshot above shows evidence of password found through Wireshark when login)

Explanation: The severity and impact of this exploit is extremely high. As you can see in the above screenshot, the password and username are shown in plaintext through packet capture. That means that attackers can easily obtain credentials through packet capture with a very low skill and low effort attack.

Prevention and Mitigation: The protocol being used is HTTP, which is not encrypted. In order to prevent this, the administrator must upgrade the HTTP protocol to HTTPS, which uses SSL/TLS to encrypt traffic. As a result, the encrypted traffic prevents malicious actors from reading sensitive data. This can be done in multiple ways. Most modern web services (i.e. GoDaddy, NamesCheap) offer HTTPS certificates which can be used.

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



```
cris...@dibart-Debian22: ~  
File Edit Tabs Help  
root@b4631fe4c03f:/var/log/nginx# ls -la  
total 8  
drwxr-xr-x 2 root root 4096 Jan 26 08:58 .  
drwxr-xr-x 1 root root 4096 Jan 26 08:58 ..  
lrwxrwxrwx 1 root root 11 Jan 26 08:58 access.log -> /dev/stdout  
lrwxrwxrwx 1 root root 11 Jan 26 08:58 error.log -> /dev/stderr  
root@b4631fe4c03f:/var/log/nginx# Cristian Di Bartolomeo 991521498
```

```
cris...@dibart-Debian22: ~/Assignment-3  
File Edit Tabs Help  
adminer_1 | [Mon Mar 14 20:56:53 2022] [::ffff:10.0.69.32]:38912 Accepted  
adminer_1 | [Mon Mar 14 20:56:53 2022] [::ffff:10.0.69.32]:38910 [302]: POST /  
adminer_1 | [Mon Mar 14 20:56:53 2022] [::ffff:10.0.69.32]:38910 Closing  
adminer_1 | [Mon Mar 14 20:56:53 2022] [::ffff:10.0.69.32]:38912 [302]: GET /?p  
gsqldb&username=postgres&db=postgres  
adminer_1 | [Mon Mar 14 20:56:53 2022] [::ffff:10.0.69.32]:38912 Closing  
adminer_1 | [Mon Mar 14 20:56:53 2022] [::ffff:10.0.69.32]:38914 Accepted  
adminer_1 | [Mon Mar 14 20:56:53 2022] [::ffff:10.0.69.32]:38914 [200]: GET /?p  
gsqldb&username=postgres&db=postgres&ns=public  
adminer_1 | [Mon Mar 14 20:56:53 2022] [::ffff:10.0.69.32]:38914 Closing  
adminer_1 | [Mon Mar 14 20:56:53 2022] [::ffff:10.0.69.32]:38916 Accepted  
adminer_1 | [Mon Mar 14 20:56:53 2022] [::ffff:10.0.69.32]:38916 [200]: GET /?p  
gsqldb&username=postgres&db=postgres&ns=public&script=db  
adminer_1 | [Mon Mar 14 20:56:53 2022] [::ffff:10.0.69.32]:38916 Closing  
adminer_1 | [Mon Mar 14 20:56:56 2022] [::ffff:10.0.69.32]:38918 Accepted  
adminer_1 | [Mon Mar 14 20:56:56 2022] [::ffff:10.0.69.32]:38920 Accepted  
adminer_1 | [Mon Mar 14 20:56:56 2022] [::ffff:10.0.69.32]:38918 [200]: GET /?p  
gsqldb&username=postgres&db=postgres&ns=public&table=authors  
adminer_1 | [Mon Mar 14 20:56:56 2022] [::ffff:10.0.69.32]:38918 Closing  
adminer_1 | [Mon Mar 14 20:56:59 2022] [::ffff:10.0.69.32]:38922 Accepted  
adminer_1 | [Mon Mar 14 20:56:59 2022] [::ffff:10.0.69.32]:38920 [200]: GET /?p  
gsqldb&username=postgres&db=postgres&ns=public&select=authors  
adminer_1 | [Mon Mar 14 20:56:59 2022] [::ffff:10.0.69.32]:38920 Closing
```

Explanation: The logs captured by the webpage are only the default Nginx logs (access.log and error.log) which appear to only be sent to stdout and stderr, meaning that they are never stored. As a result, capturing important events such as administrator changes or actions against the webpage are not easily accessible or may not even be possible. Backtracking something like an attack could be extremely hard or impossible if something like a restart happens, making this extremely severe.

Prevention and Mitigation: Adding logging functionality all depends on the service being used. In this case, Nginx is being used. To add custom logging to Nginx, you can edit Nginx's configuration file (/etc/nginx/nginx.conf) and use the "log_format" parameter. log_format will edit the default format of the logs, and you can choose the log file location for both access.log and error.log with "access_log" and "error_log" in the configuration file.

V-9: The website only uses single-factor authentication

Explanation: The website itself has only one way to authenticate the user which is by using the same password ("password123") for both admin and student.

<input type="checkbox"/> Modify	id	created_on	username	password	role
<input type="checkbox"/> edit	1	2022-03-09 02:32:18.144365+00	admin	password123	admin
<input type="checkbox"/> edit	2	2022-03-09 02:32:18.14574+00	student	password123	user

☐ Whole result

☐ 2 rows

Selected (0)

Import

← → ↻ 192.168.1.21/login.php ☆ 📄 ☰

Blog

Please sign in

Username

Password

Sign in

This blog and contents © 2022

[Back to top](#)

(Screenshot above shows no other option to authenticate the user)

Severity and Impact: The severity and impact is that its susceptible to 4 types of password attacks, 1) brute force attacks (guess all possibility) 2) dictionary attacks (try all words in English language first) 3) hybrid attacks (different variations combined) 4) rainbow table attack (uses precomputed hashes to attack).

Prevention and Mitigation: To prevent such a thing, we should use MFA (Multi-factor authentication) such as implementing something the user has (keys, debit/credit cards, security tokens, etc.), something the user knows (passwords/PIN's) or something the user is (biometrics).

Security Fix: The fix for this would be to redesign the website in such a way where we would add additional buttons in which we can further verify the user.

Bonus Mark #1 - Implement database caching using Memcached

```
yuanja@yuanja-VirtualBox:~/Desktop/Assignment-3$ sudo apt-get install memcached
php-memcached php-memcache
[sudo] password for yuanja:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libmemcached11 php-common php-igbinary php-msgpack php7.4-cli php7.4-common
  php7.4-json php7.4-opcache php7.4-phpdbg php7.4-readline
Suggested packages:
  libanyevent-perl libcache-memcached-perl libmemcached libterm-readkey-perl
  libyaml-perl php-pear
The following NEW packages will be installed:
  libmemcached11 memcached php-common php-igbinary php-memcache php-memcached
  php-msgpack php7.4-cli php7.4-common php7.4-json php7.4-opcache
  php7.4-phpdbg php7.4-readline
0 upgraded, 13 newly installed, 0 to remove and 21 not upgraded.
Need to get 4,536 kB of archives.
After this operation, 19.5 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://ca.archive.ubuntu.com/ubuntu focal-updates/main amd64 memcached am
d64 1.5.22-2ubuntu0.2 [128 kB]
Get:2 http://ca.archive.ubuntu.com/ubuntu focal/main amd64 php-common all 2:75
[11.9 kB]
Get:3 http://ca.archive.ubuntu.com/ubuntu focal-updates/main amd64 php7.4-commo
```

(Screenshot above shows installing memcached and also php memcached for php)


```
GNU nano 4.8 /etc/memcached.conf Modified
# memcached default config file
# 2003 - Jay Bonci <jaybonci@debian.org>
# This configuration file is read by the start-memcached script provided as
# part of the Debian GNU/Linux distribution.

# Run memcached as a daemon. This command is implied, and is not needed for the
# daemon to run. See the README.Debian that comes with this package for more
# information.
-d

# Log memcached's output to /var/log/memcached
logfile /var/log/memcached.log

# Be verbose
# -v

# Be even more verbose (print client commands as well)
# -vv

# Start with a cap of 64 megs of memory. It's reasonable, and the daemon default
# Note that the daemon will grow to this size, but does not start out holding
# memory
-m 256
```

(screenshot above shows editing the memcached.conf file to increase the cached from 64mb to 256 mb)

```
yuanja@yuanja-VirtualBox:~/Desktop/Assignment-3$ systemctl enable memcached.service
[sudo] password for yuanja:
● memcached.service - memcached daemon
   Loaded: loaded (/lib/systemd/system/memcached.service; enabled; vendor pre>
   Active: active (running) since Wed 2022-03-16 01:27:39 EDT; 42min ago
     Docs: man:memcached(1)
  Main PID: 16881 (memcached)
    Tasks: 10 (limit: 19125)
   Memory: 1.7M
    CGroup: /system.slice/memcached.service
            └─16881 /usr/bin/memcached -m 64 -p 11211 -u memcache -l 127.0.0.1
Mar 16 01:27:39 yuanja-VirtualBox systemd[1]: Started memcached daemon.
lines 1-11/11 (END)
```

(Screenshot above, we enable memcached.services)

Bonus Mark #3 - Allow Articles to be written using simple HTML tags & prevent the use of malicious JavaScript

First of all, the way to allow HTML tags to be written without exposing us to the vulnerabilities is to first, remove the JavaScript escape characters by calling `JSON.stringify()` function, what this essentially does is it will serialize and deserialize JavaScript objects. Second, we will escape the "<" by implementing it with

a Unicode after the serialization process is achieved. We can also apply this escaping to HTML character. However, the problem we encounter with this is that it will spoil our data, to fix this we store the information within a variable in which we call the `getElementById` and `getAttribute` to retrieve the information. Another way we can stop malicious code is to add a `nonce` attribute to prevent execution of scripts without that `nonce` attribute. Finally, the last fix would be the usage of the `<safescript>` tag. What `<safescript>` does is it will take any JavaScript code in between the `<safescript>` and carry its values as a string object.