# **Secure Coding**

אופן כתיבת קוד ע"י המתכנת שמונע פריצות, התקפות, זיבול מידע וכדומה.

**עקרונות כתיבת קוד מוגן:**

1. **Improper Error Report**

זהו מצב בו הודעות שגיאה מגיעות מה-Backend ל-Frontend כאשר ההודעות הינן הודעות מערכת.

אסור שהודעת מערכת תגיע למשתמש הסופי.

סיבה: הודעות מערכת יכולות להכיל מידע רגיש, שמות של טבלאות, שמות של עמודות, שמות של קבצים וכו'...

הגנה: הפרדה בין סביבת ה-Development בה אנו כן מדווחים את שגיאות המערכת לבין סביבת ה-Production בה אנו לא מדווחים את שגיאות המערכת אלא מדווחים הודעה כללית.

1. **Logs**

חובה לתעד בקובץ Log כל קריסה שהתרחשה.

אפשר להשתמש בספרייה מוכנה לביצוע Log, לדוגמה: <https://www.npmjs.com/package/winston>

אפשר לבנות מחלקת log משלנו.

1. **Validation**

חובה לבצע ולידציה על מידע שגוי / לא חוקי עבור POST, PUT, PATCH.

כל מאפיין שיכול להיות לגביו מידע שגוי / לא חוקי – חייב להיבדק בצורה יסודית.

1. **Too Specific Error Message**

אין לתאר שגיאה בצורה ספציפית לדוגמה עבור Login.

הודעת השגיאה: Non Existing Username אינה טובה כי זה אומר האם קיים משתמש או לא.

כנ"ל הודעת השגיאה: "הסיסמה שלך לא מתאימה לשם המשתמש. היא מתאימה למשתמש בשם איציק. האם אתה במקרה איציק?"

הודעת שגיאה נכונה תהיה "Incorrect username or password".

1. **אין להחזיר בחזרה ל-Frontend סיסמאות.**

גם לא ב-Register וגם לא ב-Login.

לפני יצירת ה-Token חובה למחוק / להסיר את הסיסמה מאובייקט ה-user שמכניסים ל-Token.

1. **Plain-Text Passwords**

אלו סיסמאות השמורות במסד הנתונים ללא שום הצפנה. שמורות as-is.

אסור.

יש לגורמים שונים גישה למסד הנתונים, לכן לא ידוע מי ראה את הסיסמה של היוזר ויכול להשתמש איתה.

הטכניקה אינה הצפנה (Encrypt) ופענוח (Decrypt) של סיסמאות.

עבור סיסמאות מבצעים פעולה הנקראת גיבוב (Hashing).

זו פעולה שיוצרת מחרוזת מעורבלת ממחרוזת מקורית, אך שלא ניתן לפענח את המחרוזת המעורבלת ולחזור למחרוזת המקורית (אלגוריתם חד כיווני).

מה שחובה עבור אלגוריתם כזה הוא שמאותה המחרוזת תמיד יווצר אותו ה-Hash.

יש כל מיני אלגוריתמי Hash:

* SHA1
* SHA256
* SHA512
* MD5
* עוד...

נוהל פעולה:

שלב ה-Register:

“1234” 🡪 Hash 🡪 “slkdfjslkdjsldfjslkdfjsldfjsldfjsdlfj” 🡪 Save in DB

שלב ה-Login:

“1234” 🡪 Hash 🡪 “slkdfjslkdjsldfjslkdfjsldfjsldfjsdlfj” 🡪 Compare this Hash to existing Hash in DB

1. **Non-Salted Hashed Passwords**

עבור סיסמאות פשוטות מידי, ניתן לגלות מה הסיסמה בהינתן ה-Hash ע"י אתרים שבדקו מיליארדי סיסמאות והפכו אותן ל-Hash – הם פשוט מחפשים את ה-Hash במאגר ורואים איזו סיסמה יצרה אותו. (הטבלאות הללו נקראות Rainbow Tables).

לכן אסור לבצע Hash לסיסמה פשוטה.

* בצד ה-Front אין לאפשר למשתמש להכניס סיסמה פשוטה. יש לחייב מינימום תווים (לרוב 6), אות גדולה, אות קטנה, סיפרה, תו סימן, אות לטינית...
* גם ב-Backend אפשר לבדוק מבחינת ולידציה את אותה החוקיות.
* יש לבצע גם Salt לסיסמה (להמליח אותה) – לפזר בין התווים שלה תווים אחרים כלשהם וכך ליצור סיסמה חזקה שבוודאות לא קיימת ב-Rainbow Tables ועליה לבצע Hash.

דוגמה לשימוש:

Salt: “abcd1234”

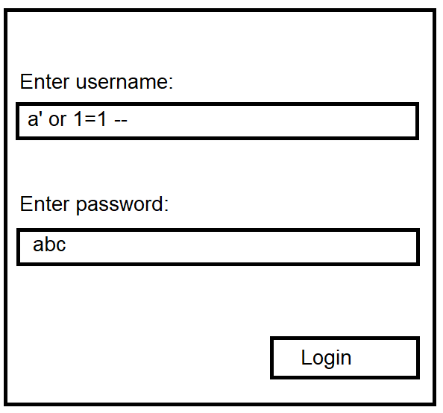
Plaint-Text Password: “HelloStudent”

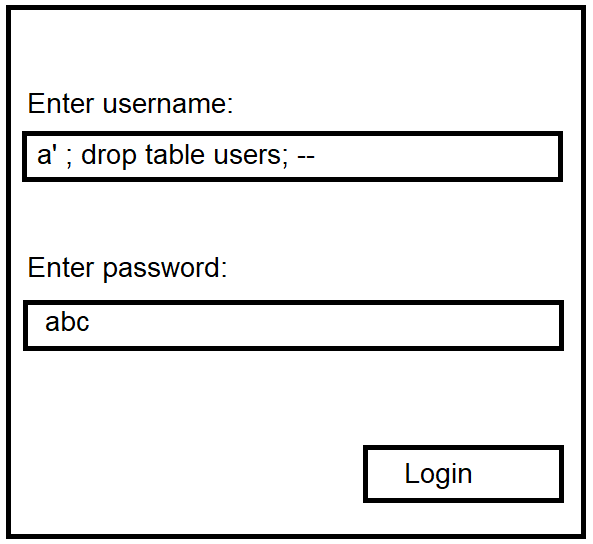
Salted Password: “HeallboSctudde1nt234”

Hash performed on “HeallboSctudde1nt234”

1. **עבור Production – מסד נתונים חייב להיות מוגן ע"י user ו-password.**
2. **פרצת SQL-Injection**

האקר ע"י תיבת קלט כלשהי (תיבת טקסט לדוגמה) מכניס קטע SQL שמשולב בשאילתה המקורית ומשנה את המשמעות שלה כך שניתן לפרוץ, למחוק, לשנות, להרוס וכו':





פתרון: ביצוע execute של שאילתת SQL ע"י סימני שאלה בתוך השאילתה, כך שהמערכת תכניס את הערכים במקום סימני השאלה. יש לכן לשלוח ל-DAL שאילתה + מערך של ערכי ה-values, אותו יש לשלוח לפונקציית ה-query.

1. **XSS Attack**

XSS – Cross Site Scripting

זהו מקרה בו הודעה הנשלחת לשרת שאמורה להיות מוצגת על דפדפן בנויה כקוד HTML או סקריפט JavaScript.

במצב כזה הדפדפן יציג את ה-HTML או יריץ את ה-JS.

לדוגמה, הודעה הנשלחת לפורום המכילה את המחרוזת הבאה: <h1>Hi</h1>

דוגמה נוספת: ההודעה: <script>window.location=<http://somesite.com></script>

פתרונות:

1. בצד השרת – הסרת התגיות ע"י ספרייה כגון npm i striptags
2. בצד הלקוח – הצגת מידע ממסד נתונים כדוגמת פורום בתגית המציגה את המידע כמו שהוא, לדוגמה <![CDATA[ … ]]> או תגית מקבילה.
3. **IDOR**

Insecure Direct Object Reference

זהו מצב בו האקר רואה ערך שניתן לשלוח לשרת לצורך קבלת מידע או ביצוע פעולות והוא משנה את הערך הזה לערך אחר שמביא מידע חסוי אחר או מבצע פעולות אחרות.

פתרון: הסתרת מידע שניתן לשינוי בקלות. לדוגמה, אם id שהינו מספר רץ מוצג וניתן לשינוי בכדי לקבל מידע – יש להחליפו ל-UUID. לדוגמה, לא לאפשר להכניס נתונים לצורך קבלת מידע לפיהם (שמות קבצים, שמות טבלאות...)

1. **DoS Attack**

DoS – Deny of Service

זוהי התקפה של האקר אחד או יותר על אתר.

כל מה שההתקפה עושה זה גלישות מאסיביות לאתר, לדוגמה ע"י לולאה שפונה ללא הפסקה לאתר לקבלת מידע.

ברגע שזה קורה, האתר משרת את הפניות הפיקטיביות הללו. ואז הוא לא יכול לשרת לקוחות אמיתיים בצורה טובה/מהירה וכו'. בסופו של דבר האתר גם יכול לקרוס.

פתרון: שימוש בספרייה שמחשבת כמה פניות מתבצעות מאותו IP Address ומונעות פניות מופרזות בתוך חלון זמן מוגדר.

דוגמה לספרייה כזו: npm i express-rate-limit

1. **DDoS**

Distributed Deny of Service

ביצוע DoS Attack אך בכל Request משנים את ה-IP Address לרנדומלי.

פתרון: אין.

1. **BOT Attack**

קיצור של המילה Robot

ביצוע Request-ים אוטומטית ע"י "רובוט" – תוכנה או סקריפט שבעיקר מבצעים הוספה של מידע למסדי נתונים.

לדוגמה, יצירת חשבונות פיקטיביים.

פתרונות:

CAPTCHA – Completely Automated Public Turing Test to tell Computers and Humans Apart

reCAPTCHA – זהו AI של גוגל שבנוי מסקריפט (JavaScript) ששמים על הדף שבו רוצים למנוע BOT, והוא בודק את התנהגות היוזר על אותו הדף ולפי ההתנהגות יודע האם היוזר הוא אדם או BOT.

1. **SSL**

Secure Socket Layer

שכבה נוספת על HTTP של הצפנת ה-Request וה-Response.

פרוטוקול התקשורת נקרא https.

כתובת http אינה מוצפנת – ניתן לקרוא כל Request וכל Response אם מאזינים לתקשורת שלכם

(נקרא Man-in-the-Middle)

אתרים צריכים להיות https ולא על http.

לצורך כך אנו צריכים תעודה דיגיטלית המשמשת להצפנה. תעודה דיגיטלית שייכת ספציפית לאתר מסוים.

מי שמנפיק תעודות כאלו זה אתרים בינלאומיים שיש להם את הסכמות הזו. אתרי CA – Certificate Authority.

דוגמה לאתר חינאמי כזה: <https://letsencrypt.org>